Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Уральский государственный педагогический университет»
Институт математики, информатики и информационных технологий
Кафедра информационно-коммуникационных технологий в образовании

ТЕХНОЛОГИЯ РАЗРАБОТКИ ИГРОВЫХ ПРИЛОЖЕНИЙ ДЛЯ ОС ANDROID

Выпускная квалификационная работа
бакалавра по направлению подготовки
09.03.02 – Информационные системы и технологии

Исполнитель: студент группы БС-41z
Института математики, информатики и ИТ
Галиев Р.Р.

Руководитель: доцент кафедры ИИТиМОИ
Газейкина А.И.

Работа допущена к защите
«____»_____ 2016 г.
Зав. кафедрой _____

Екатеринбург – 2016

Реферат

Галиев Р.Р. ТЕХНОЛОГИЯ РАЗРАБОТКИ ИГРОВЫХ ПРИЛОЖЕНИЙ ДЛЯ ОС ANDROID, выпускная квалификационная работа: 74 стр., рис. 31, табл. 0, библ. 24 назв.

Ключевые слова: РАЗРАБОТКА ИГРОВЫХ ПРИЛОЖЕНИЕ ДЛЯ ОС ANDROID, РАЗРАБОТКА ПРИЛОЖЕНИЙ.

Объект разработки – игровое приложение для ОС Android.

Цель работы – разработать игровое приложение для ОС Android с использованием библиотеки libGDX, с последующей публикацией игры в Google Play. Анализировать полученные результаты, после публикации игры, с помощью Google Play Developer Console.

В работе описаны этапы разработки игрового приложения для ОС Android, публикация в Google Play, анализ отзывов и числа установок в Google Play Developer Console, анализ доходов с рекламных площадок AdMob и Chartboost.

В ходе работы было создано мобильное приложение «Проект АРсИИ: Эксперимент» с использованием IDE Android Studio. Исходные коды игрового приложения написаны на языке Java с использованием библиотек: LibGDX для разработки графических приложений; расширенные библиотеки Android SDK - Google play service для внедрения игровых сервисов Google Play Game Service (рейтинг игроков, достижения) и рекламного баннера AdMob; Chartboost SDK для реализации просмотра видео-рекламы с вознаграждением.

Игра опубликована в Google Play и доступна бесплатно в 10 странах:

- в Австралии, Великобритании, Индии, Канаде, США, Филиппинах на английском языке;

- в России, Казахстане, Белоруссии, Украине на русском языке.

Оглавление

Введение

Android – универсальная операционная система, на которой работает более миллиарда устройств: от смартфонов и планшетов до часов и даже автомобилей. Это самая популярная мобильная ОС в мире. Android - это платформа мирового класса для создания приложений и игр для пользователей ОС Android везде, а также открытый рынок для распространения своих игр и приложений по всему миру. Открытость и доступность Android сделала его фаворитом для потребителей и разработчиков одинаково, стимулировав устойчивый рост в потреблении приложения. Пользователи Android загружают миллионы приложений и игр в Google Play каждый месяц. Большая доля загрузок приходится именно на игровые приложения.

Такая популярность игровых приложений неспроста. На данный момент игровая индустрия становится такой же популярной, как и кино, которое начало обретать популярной в XX веке. В начале XXI века ведутся споры о том, что видео игры - это как отдельная область искусства, так же как и кино, литература, живопись, музыка.

В связи вышесказанным, **объектом** исследования будет являться процесс разработки игровых приложений работающих на смартфонах и планшетах под управлением операционной системы Android. **Предмет** исследования: технология разработки игровых приложений для операционной системы Android с использованием библиотеки libGDX.

Цель исследования: разработать игровое приложение для операционной системы Android с использованием библиотеки libGDX, с последующей публикацией игры в Google Play. Анализировать полученные результаты, после публикации игры, с помощью Google Play Developer Console.

Задачи:

1. Произвести анализ особенности разработки приложения для операционной системы Android.

2. Произвести анализ и обосновать выбор технологии разработки приложений в IDE Android Studio с использованием библиотеки libGDX.
3. В соответствии с техническим заданием провести разработку игрового приложения «Проект АРсИИ: Эксперимент» для операционной системы Android.
4. Опубликовать разработанное игровое приложение в Google Play и собрать статистические данные по приложению из: Google Play Developer Console, AdMob и Chartboost.

Глава 1. Теоретические основы разработки приложений для мобильных устройств на базе операционной системы Android

В этой главе будет рассмотрено, что представляет собой ОС Android, и некоторые инструменты, существующие для разработки игровых приложений для данной платформы.

1.1 Операционная система Android

Первое громкое упоминание Android прозвучало в 2005 году, когда Google приобрел маленький стартап-проект Android, Inc. Это действие вызвало множество спекуляций на тему выхода Google на мобильный рынок. Конец слухам положил в 2008 году релиз Android 1.0, после чего Android стала новым игроком на самом перспективном рынке. С тех пор идет конкуренция между ней и уже устоявшимися платформами вроде iOS (ранее известной как iPhone OS), Blackberry и Windows Phone.

Поскольку Android — проект с открытым исходным кодом, барьер вхождения для производителей устройств довольно низок. Они могут выпускать аппараты для любых ценовых сегментов и модифицировать систему для ее лучшего взаимодействия с конкретными устройствами. Таким образом, область применения Android не ограничивается смартфонами высшего ценового уровня, что делает ее потенциальную аудиторию весьма широкой.

Ключевой составляющей успеха Android стало образование Open Handset Alliance (OHA) в конце 2007 года. В настоящий момент в OHA входят 86 различных компаний такие, как HTC, Qualcomm, Motorola, NVIDIA, LG, T-Mobile, Acer, ASUS, Dell, Huawei, Samsung, ARM, Intel, MediaTek и многие другие, которые сотрудничают в разработке открытых стандартов для мобильных устройств. Хотя ядро Android создано в основном в Google, все члены OHA вносят в него свой вклад в той или иной форме.

Android, по сути, представляет собой мобильную операционную систему (ОС) и платформу, основанную на ядре Linux версии 2.6, свободную для использования в коммерческих и некоммерческих целях. Многие члены OHA собирают собственные версии Android для своих устройств с измененным пользовательским интерфейсом (User Interface, UI), например HTC Sense или Motorola MOTOBLUR. Открытая природа Android позволяет фанатам делать собственные сборки. Они обычно называются модами, прошивками или ROM. Самая известная прошивка на данный момент разработана парнем, известным как Cyanogen, и предназначена для оснащения новейшими улучшениями всех Android-устройств. [2]

Каждой новой версии Android присваивалось название, соответствующее названию какого-либо десерта. Версия Android 2.2, которая также называется «Froyo» (замороженный йогурт), вышла в мае 2010 года. В ней появилась поддержка внешней памяти, позволявшая хранить приложения на внешнем устройстве (вместо внутренней памяти устройства Android). С помощью службы C2DM (Android Cloud to Device Messaging, обмен сообщениями с устройствами через облако) разработчики приложений могут использовать программы и данные, хранящиеся в «облаке» — то есть на удаленных компьютерах (серверах) в Интернете, вместо того чтобы хранить их на настольном компьютере, ноутбуке или мобильном устройстве. Облачные технологии предоставляют гибкие средства наращивания или сокращения вычислительных ресурсов под конкретные потребности в любой момент времени. Это делает их более экономически эффективными по сравнению с приобретением дорогостоящего оборудования, гарантирующего достаточную вычислительную мощность и емкость памяти для периодических пиковых нагрузок. Android C2DM позволяет разработчикам приложений пересылать данные со своих серверов на приложения, установленные на устройствах Android, даже если эти приложения в данный момент не активны. Сервер оповещает приложения, предлагая им подключиться к нему для приема обновления или пользовательских данных. Сейчас технология C2DM

считается устаревшей — на смену ей пришла технология Google Cloud Messaging.

Версия Android 2.3 Gingerbread («имбирный пряник»), появившаяся в декабре 2010 года, предлагает новые возможности для пользователя — более удобную клавиатуру, улучшенные навигационные функции, более эффективное использование батареи и ряд других преимуществ. Также в ней добавились новые средства разработчика для передачи данных (например, технологии, упрощающие телефонные звонки и ответы на них в приложениях), мультимедийные технологии (новые API для работы со звуком и графикой) и игровые средства (повышение быстродействия и новые датчики — например, гироскоп для обработки перемещений в пространстве).

Одним из самых значительных нововведений Android 2.3 стала поддержка NFC (Near-Field Communication) — технологии беспроводной высокочастотной связи малого радиуса действия, которая дает возможность обмена данными между устройствами, находящимися на расстоянии нескольких сантиметров. Уровень поддержки и функциональность NFC зависит от конкретного устройства. NFC может использоваться для электронных платежей (например, когда вы прикасаетесь устройством Android с поддержкой NFS к платежному блоку на торговом автомате), передачи данных (контактов, фотографий и т. д.), сопряжении устройств и аксессуаров и т. д.

В версии Android 3.0-3.2 Honeycomb («пчелиные соты») появились усовершенствования пользовательского интерфейса, предназначенные для устройств с большим экраном (то есть планшетов): усовершенствованная клавиатура для более эффективного ввода данных, трехмерный пользовательский интерфейс, упрощение переходов между экранами в приложении и ряд других улучшений. Некоторые новые средства разработчика Android 3.0:

- фрагменты, описывающие части пользовательского интерфейса приложения, которые могут объединяться в один экран или использоваться на разных экранах;

- постоянная панель действий в верхней части экрана, предоставляющая дополнительные средства для взаимодействия с приложениями;
- возможность добавления макетов для большого экрана к существующим приложениям, рассчитанным на малые экраны, чтобы оптимизировать приложение для разных вариантов размера экрана;
- привлекательный и более функциональный пользовательский интерфейс «Holo» имитирующий «голографический» эффект;
- усовершенствованные графические и мультимедийные возможности;
- возможность использования многоядерных процессоров для повышения быстродействия;
- улучшенная поддержка Bluetooth (например, возможность проверки подключенных устройств — гарнитуры, клавиатуры и т. д.);
- фреймворк для анимации пользовательского интерфейса или графических объектов.

Версия Android 4.0 Ice Cream Sandwich («сэндвич с мороженым»), вышедшая в 2011 году, объединила Android 2.3 (Gingerbread) и Android 3.0 (Honeycomb) в одну операционную систему, предназначенную для использования на всех устройствах Android. Эта версия Android позволяет включить функции, поддерживаемые в версии Honeycomb и ранее доступные только для планшетов («голографический» интерфейс пользователя, новый лаунчер и т. д.), в приложения для смартфонов. Так обеспечивается простое масштабирование приложений, позволяющее использовать их на различных устройствах. В версии Ice Cream Sandwich также появился ряд новых API для улучшения обмена данными между устройствами, технологий для пользователей с ограниченными возможностями (например, с ослабленным зрением), поддержки социальных сетей и т.д.

Версия Android Jelly Bean («жевательная конфета»), выпущенная в 2012 году, включает поддержку внешних экранов, усовершенствованные средства безопасности, улучшенное оформление (например, виджеты приложений с из-

меняемыми размерами и крупные оповещения) и функции более плавного переключения между приложениями и экранами.

Версия Android 4.4 KitKat, выпущенная в октябре 2013 года, включает ряд усовершенствований, обеспечивающих работу операционной системы на любых устройствах Android, включая старые устройства с ограниченной памятью, особенно популярные в развивающихся странах. Переход большего количества пользователей на KitKat сократит «фрагментацию» версий Android на рынке. Фрагментация создавала проблемы для разработчиков, которые были вынуждены проектировать приложения для разных версий операционной системы или ограничивать круг потенциальных пользователей, разрабатывая приложение для конкретной версии операционной системы. [1]

Android KitKat также включает усовершенствования безопасности и доступности, улучшенные графические и мультимедийные возможности, средства анализа памяти и т.д. Так же в данной версии появилась новая среда выполнения для приложений ART (Android RunTime) в качестве экспериментальной (его можно активировать через настройки для разработчика).

Главным преимуществом ART перед Dalvik - улучшенное энергопотребление. При установке Java-код приложения сразу компилируется в машинный код (AOT-компиляция), тогда как Dalvik компилировал Java-код в свой байткод dex (Dalvik executable), а после запуска программы компилировал его в машинный код в реальном времени (JIT-компиляция). Поэтому ART экономит энергию. Правда, это происходит за счёт увеличения используемого пространства и замедления инсталляции приложений. Google уверяет, что замедление не критическое. Инновации вроде ART стали возможны благодаря увеличению объёма памяти в современных смартфонах. ART полностью совместим с байткодом Dalvik, так что с точки зрения разработчиков ничего не изменилось: можно писать приложения такие же, как раньше. Новая среда при установке приложения создаёт исполняемый файл формата ELF с машинным кодом. В ART значительно улучшена работа сборщика мусора, из-за постоянных включений кото-

рого раньше графический интерфейс «подтормаживал», замирая на доли секунды. Теперь паузы на сборку мусора сокращены практически до нуля: вместо десятков или сотен миллисекунд большинство операций сборщика мусора теперь завершается примерно за 1 миллисекунду или чуть больше. [3]

Версия 5.0 (Lollipop – «Леденец») предназначена для различных платформ – от часов и планшетов до телевизоров и автомобилей. Данная версия отличается простым и красочным интерфейсом, выполненным в лучших традициях Material Design. Обычно, когда какая-нибудь компания внедряет функции для улучшенного энергопотребления, речь идёт о небольших оптимизациях. Время работы от аккумуляторов увеличивают на несколько процентов. В случае с Android 5.0 (где работает ART и сделаны другие улучшения) всё обстоит современно иначе. Независимые тесты показывают, что у Nexus 5 после апгрейда ОС время работы увеличилось с 345 до 471 минуты, то есть на 36%! Тест включал в себя бесконечное обновление веб-страницы в браузере при включенном WiFi. Правда, есть вероятность, что экономия настолько велика только в случае интенсивного использования смартфона/планшета, когда ART проявляет себя во всей красе. Если же смартфон ничего не делает с включенным дисплеем (например, в режиме чтения книги), то время работы от аккумулятора, наверное, не сильно изменится. Заметная разница произошла в производительности приложений. Большинство программ демонстрируют прибавку в скорости 50-80%, а некоторые ускоряются в два-три раза.

На момент написания к выходу готовилась версия 7.0 (Nougat), которая доступна для разработчиков для тестирования, но актуальной является версия 6.0 (Marshmallow) в которой появилась возможность контролировать доступ приложений к личным данным, камере, памяти и другим параметрам устройства. Благодаря функциям Doze и App Standby новая версия экономит заряд батареи для самых важных задач. [7]

Ниже (Рисунок 1.1.1) приводиться статистика на 5 сентября 2016 года, которая показывает соотношение количества устройств работающих на какой либо версии ОС Android. [11]

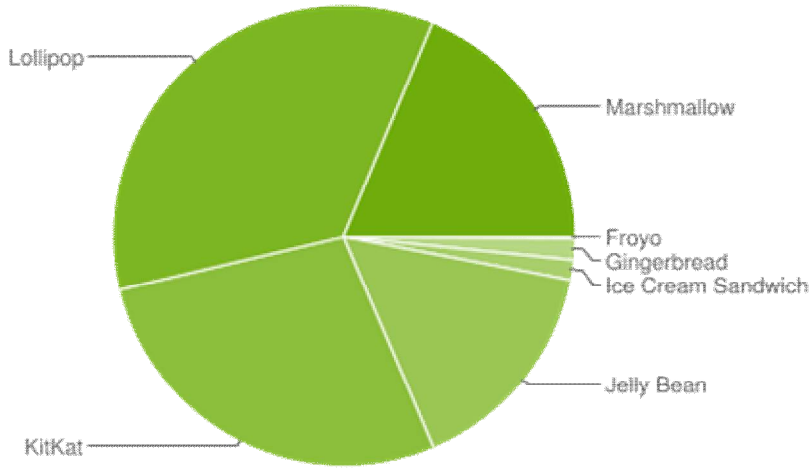


Рисунок 1.1.1. Самыми распространёнными версиями являются KitKat, Lollipop и Marshmallow

Android — это не просто еще один дистрибутив Linux для мобильных устройств. При разработке для Android разработчику, скорее всего, не придется иметь дело с сами ядром Linux. С точки зрения программиста, Android — платформа, абстрагирующая разработчика от ядра и позволяющая ему создавать код на Java. Android обладает несколькими полезными возможностями. Во-первых, это фреймворк, предлагающий большой набор API для создания различных типов приложений и, кроме того, обеспечивающий возможности повторного использования и замены компонентов, которые предлагаются платформой и сторонними приложениями. Во-вторых, наличие виртуальной машины Dalvik, отвечающей за запуск приложений на Android. Кроме того, к услугам разработчика набор графических библиотек для 2D- и 3D-приложений, поддержка мультимедиа-форматов (Ogg Vorbis, MP3, MPEG-4, H.264, PNG), API для доступа к камере, GPS, компасу, акселерометру, сенсорному экрану, джойстику и клавиатуре. Имеется даже специальное API для воспроизведения фоновых звуковых эффектов. Не все Android-устройства обладают всеми этими возможностями — причиной является аппаратное разделение. Конечно, список возможностей An-

droid не исчерпывается выше сказанным. Однако для разработки игр они будут наиболее важны.

Архитектура Android формируется из набора компонентов. Каждый компонент построен на основе элементов более низкого уровня. Далее представлен краткий обзор главных компонентов Android.

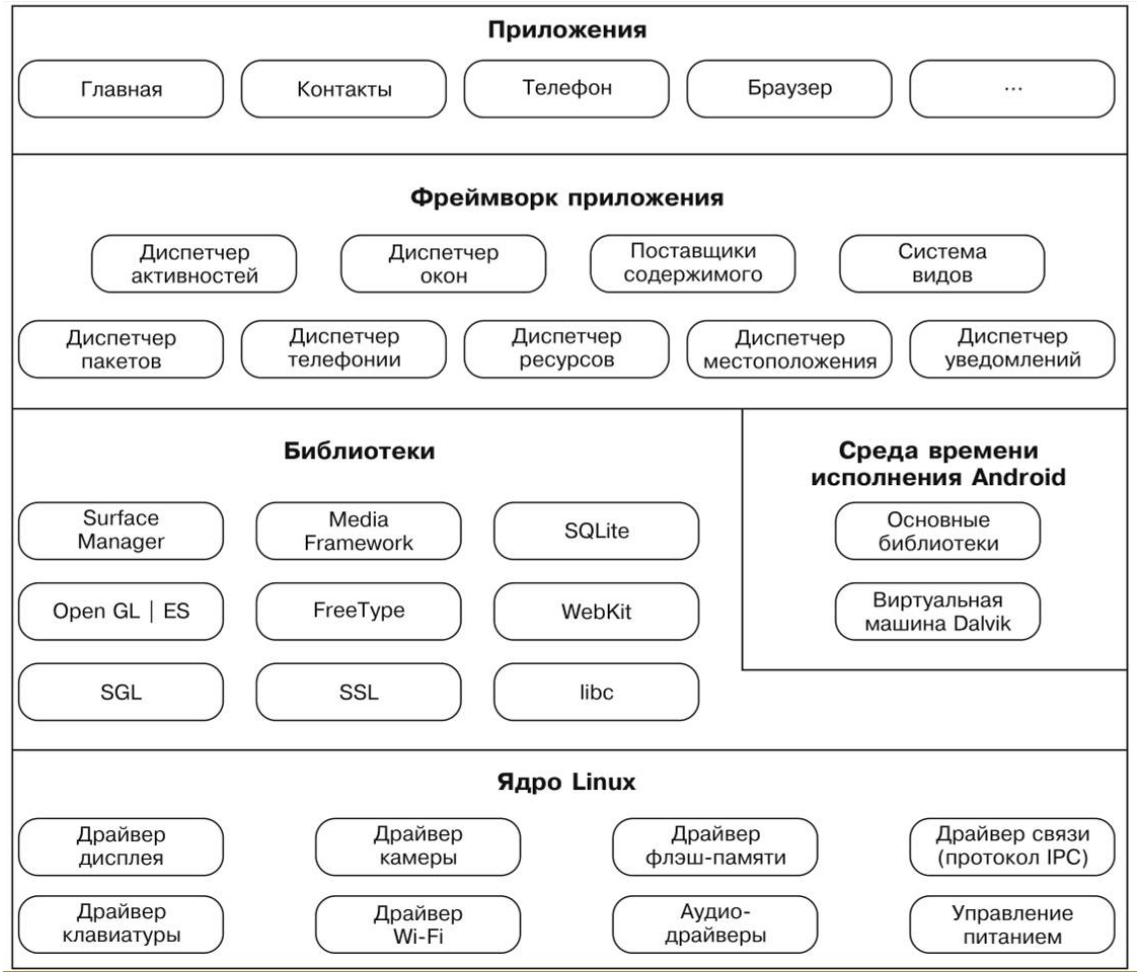


Рисунок 1.1.2. Архитектура ОС Android

В нижней части рисунка (Рисунок 1.1.2) вы можете увидеть, что ядро Linux предлагает основные драйверы для аппаратных компонентов системы. Кроме того, ядро отвечает за память, управление процессами, поддержку сети и т. д. Среда выполнения Android, являющаяся надстройкой над ядром, отвечает за порождение и выполнение приложений Android. Каждая программа работает в собственном процессе со своей виртуальной машиной Dalvik. Помимо библиотек ядра, предлагающих некоторую функциональность Java SE, существует

также набор родных библиотек на C/C++, создающих основу для фреймворка приложения (расположенного на уровень выше, чем библиотеки Рисунок 1.1.2). Эти системные библиотеки в большинстве своем отвечают за сложные в вычислительном смысле задачи (прорисовка графики, воспроизведение звука, доступ к базе данных), не очень подходящие для виртуальной машины Dalvik. API в них обернуты с помощью классов Java во фреймворк приложения. Фреймворк приложения связывает вместе системные библиотеки и среду выполнения, создавая таким образом пользовательскую сторону Android. Фреймворк управляет приложениями и предлагает продуманную среду, в которой они работают. Разработчики создают приложения для этого фреймворка с помощью набора программных интерфейсов на Java, охватывающих такие области, как разработка пользовательского интерфейса, фоновые службы, оповещения, управление ресурсами, доступ к периферии и т. д. Все ключевые приложения, поставляемые вместе с ОС Android (например, почтовый клиент), написаны с помощью этих API.

Приложения, будь они с интерфейсом или с фоновыми службами, могут связываться с другими приложениями. Эта связь позволяет одному приложению использовать компоненты других. Простой пример — программа, делающая фотоснимок и потом обрабатывающая его. Приложение запрашивает у системы компонент другого приложения, обеспечивающий это действие. Далее первое приложение может повторно использовать этот компонент (например, от встроенного приложения камеры или от фотогалереи). Подобный алгоритм снимает значительную часть ноши с программиста, а также позволяет настроить многообразие аспектов поведения Android.

Google Play (раньше назывался Android Market) был открыт для публики в октябре 2008 года компанией Google. Это онлайн-магазин приложений, позволяющий пользователям находить и устанавливать сторонние приложения. Магазин доступен как с помощью приложения на телефоне, как и с использо-

ванием браузера на компьютере. Магазин позволяет независимым разработчикам предлагать свои приложения бесплатно или за деньги.

Пользователи могут получить доступ к магазину после регистрации учетной записи Google. Покупатель вправе вернуть приложение в течение 15 минут после покупки, чтобы получить свои деньги обратно. Ранее время возврата составляло 24 часа, и столь резкое его уменьшение не слишком обрадовало пользователей. Программисту, чтобы публиковать свои приложения, необходимо зарегистрироваться в Google в качестве разработчика Android. Это будет стоить ему $25 единовременно. Уже через несколько минут после успешной регистрации становится возможным публиковать свои творения.

В Google Play отсутствует процесс утверждения опубликованного приложения, но имеется система разрешений. Пользователю перед установкой демонстрируются разрешения, которые необходимо предоставить приложению для его работы. Эти разрешения включают в себя доступ к телефонным службам, сети, карте памяти и т. д. Установка приложения совершается только после одобрения пользователем этих разрешений. Система полагается на пользователя в данном вопросе. Надо сказать, что для настольных приложений (особенно это касается Windows-систем) такой подход работает не слишком хорошо. Что касается Android — пока это эффективная мера; всего несколько приложений были удалены с Google Play по причине их вредоносного поведения. [2]

Для разработки игровых приложений под операционную систему Android можно использовать стандартный инструмент для разработки приложений Android Studio. Так же существует несколько коммерческих и открытых игровых движков и фреймворков, такие как Unity, Unreal engine, Torque2D, Cocos 2D-X, libGDX, Xamarin, Corona SDK.

Фреймворк отличается от игрового движка тем, что он позволяет контролировать разработчику каждый аспект среды разработки игр. Правда, разработчику приходится разбираться, как именно решить ту или иную задачу (напри-

мер, как организовать игровой мир, как обрабатывать экраны и переходы и т. д.).

Игровой движок, с другой стороны, ориентирован на специфические задачи. Он диктует разработчику, как он должен решать те или иные проблемы, предоставляя простые в использовании модули для типичных задач и общую архитектуру разрабатываемой игры. В некоторых движках можно создавать игры практически не прибегая к программированию.

Но игровые движки не являются панацей для создания игр, в них тоже есть свои минусы, причем некоторые из них довольно критичны. Например, исходный размер игры, после установки на устройство, будет занимать больше памяти, что является весьма не приятным для обладателей бюджетных устройств, где память для пользователя весьма ограничена. Разрабатываемая игра может не соответствовать готовым решениям, присутствующие в игровом движке. И часто приходится самостоятельно модифицировать движок для того, чтобы достичь своих целей, что может быть невозможно, если его исходный код для разработчика недоступен. Движки могут значительно сократить время разработки, но могут и увеличить его в том случае, если разработчик встретит проблему, которая не была предусмотрена создателями игрового движка.

Так же можно отметить и тот факт, что не все игровые движки являются абсолютно бесплатными, многие из них требуют от разработчика либо покупать игровой движок и прилагаемый ему инструментарий за большую сумму денег, либо отчислений от прибыли полученной разработчиком создавший игру на данном движке.

Выбирая между фреймворком и движком, основываются на персональных предпочтениях, бюджете и целях. При работе над данным проектом, были использованы IDE Android Studio и фреймворк LibGDX. Далее будут рассмотрены именно эти инструменты.

1.2 Обзор инструментов для разработки

1.2.1 Android Studio

Android Studio — это интегрированная среда разработки (IDE) для работы с платформой Android, анонсированная 16 мая 2013 года на конференции Google I/O.

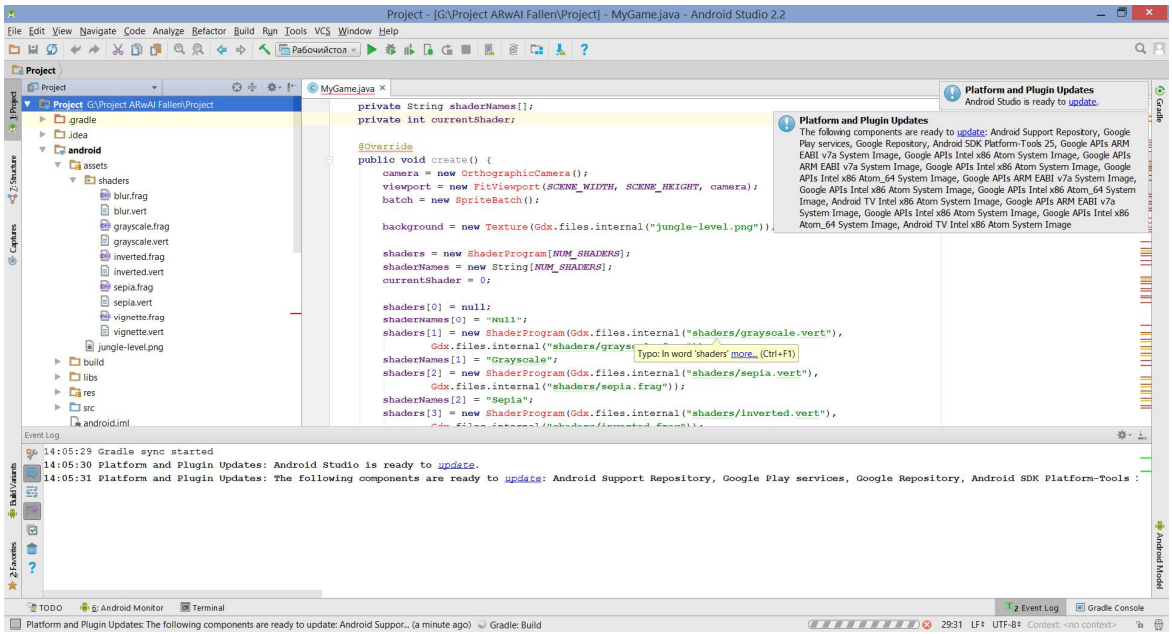


Рисунок 1.2.1. Внешний вид IDE Android Studio

IDE находится в свободном доступе начиная с версии 0.1, опубликованной в мае 2013, затем перешла в стадию бета-тестирования, начиная с версии 0.8, которая была выпущена в июне 2014 года. Первая стабильная версия 1.0 была выпущена в декабре 2014 года, тогда же прекратилась поддержка плагина Android Development Tools (ADT) для Eclipse. [10]

Android Studio, основанная на программном обеспечении IntelliJ IDEA от компании JetBrains, официальное средство разработки Android приложений. Данная среда разработки доступна для Windows, OS X и Linux.

Новые функции появляются с каждой новой версией Android Studio. На данный момент доступны следующие функции:

- Расширенный редактор макетов: WYSIWYG, способность работать с UI компонентами при помощи Drag-and-Drop, функция предпросмотра макета на нескольких конфигурациях экрана.
- Сборка приложений, основанная на Gradle.
- Различные виды сборок и генерация нескольких .apk файлов
- Рефакторинг кода
- Статический анализатор кода (Lint), позволяющий находить проблемы производительности, несовместимости версий и другое.
- Встроенный ProGuard и утилита для подписывания приложений.
- Шаблоны основных макетов и компонентов Android.
- Поддержка разработки приложений для Android Wear и Android TV.
- Встроенная поддержка Google Cloud Platform, которая включает в себя интеграцию с сервисами Google Cloud Messaging и App Engine.
- Android Studio 2.1 поддерживает Android N Preview SDK, а это значит, что разработчики смогут начать работу по созданию приложения для новой программной платформы.
- Новая версия Android Studio 2.1 способна работать с обновленным компилятором Jack, а также получила улучшенную поддержку Java 8 и усовершенствованную функцию Instant Run.
- Platform-tools 23.1.0 для Linux без объявления стала исключительно 64-разрядной, даже при попытке установить 32-разрядную версию. Иными словами Android Studio больше НЕ работает (выдаёт неустранимые ошибки) в 32-разрядных версиях Linux.

Каждый проект в Android Studio содержит один или несколько модулей с файлами исходного кода и файлами ресурсов. Типы модулей включают:

- Модули приложения для Android
- Модули библиотеки
- Модули механизма приложения Google

По умолчанию Android Studio выводит на экран файлы проекта в представлении проекта Android (Рисунок 1.2.2). Это представление организовано модулями, чтобы обеспечить быстрый доступ к файлам ключевого источника проекта.

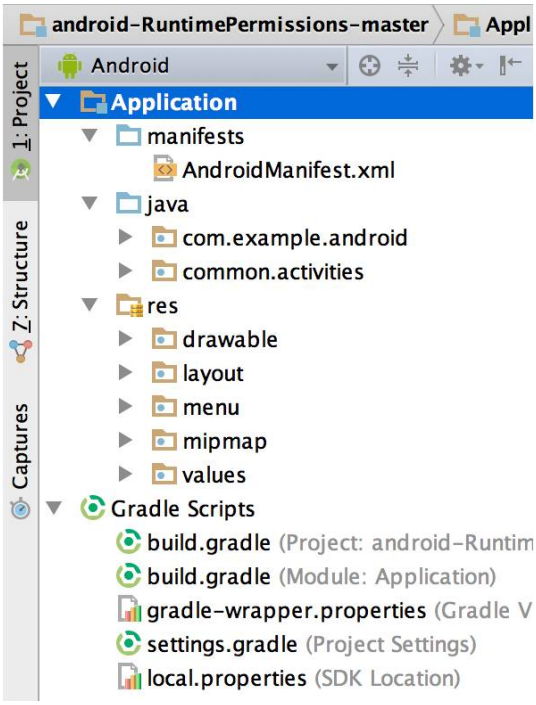


Рисунок 1.2.2. Вид проекта

Все файлы типа "build" видимы на верхнем уровне в соответствии со сценариями Gradle, и каждый модуль приложения содержит следующие папки:

- manifests: Содержит файл AndroidManifest.xml.
- java: Содержит файлы исходного кода Java, включая тестовый код JUnit.
- res: Содержит все ресурсы некода, такие как разметки XML, строки UI (пользовательский интерфейс) и растровые изображения.

Структура проекта Android на диске отличается от представления в IDE. Чтобы видеть фактическую файловую структуру проекта, во вкладке Project нужно выбрать из выпадающего списка Project.

Есть возможность настроить представление о файлах проекта, чтобы фокусироваться на определенных аспектах разработки приложений. Например,

выбор проблемного представления проекта, выводит на экран ссылки к исходным файлам, содержащим любое не правильное кодирование и синтаксические ошибки, такие как недостающий закрывающий тэг элемента XML в файле расположения (Рисунок 1.2.3).

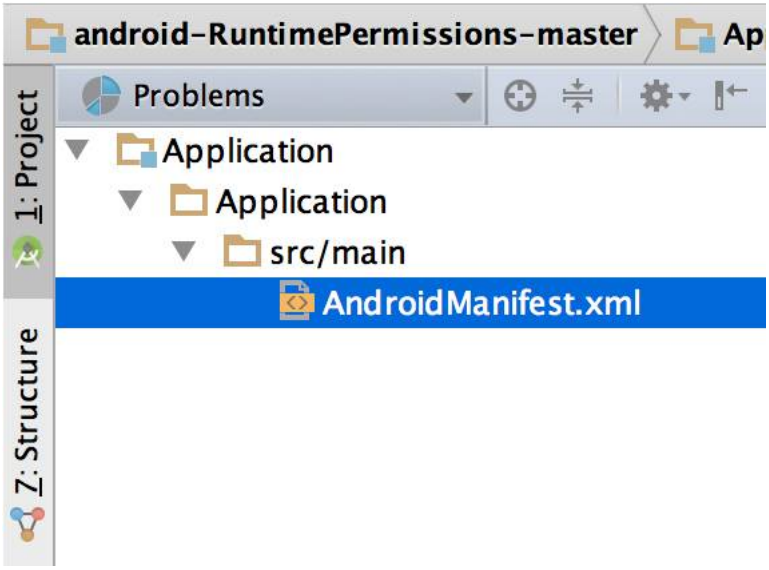


Рисунок 1.2.3. Вкладке Problems находятся файлы, в которых присутствует ошибка.

Главное окно Android Studio составлено из нескольких логических областей (Рисунок 1.2.4).

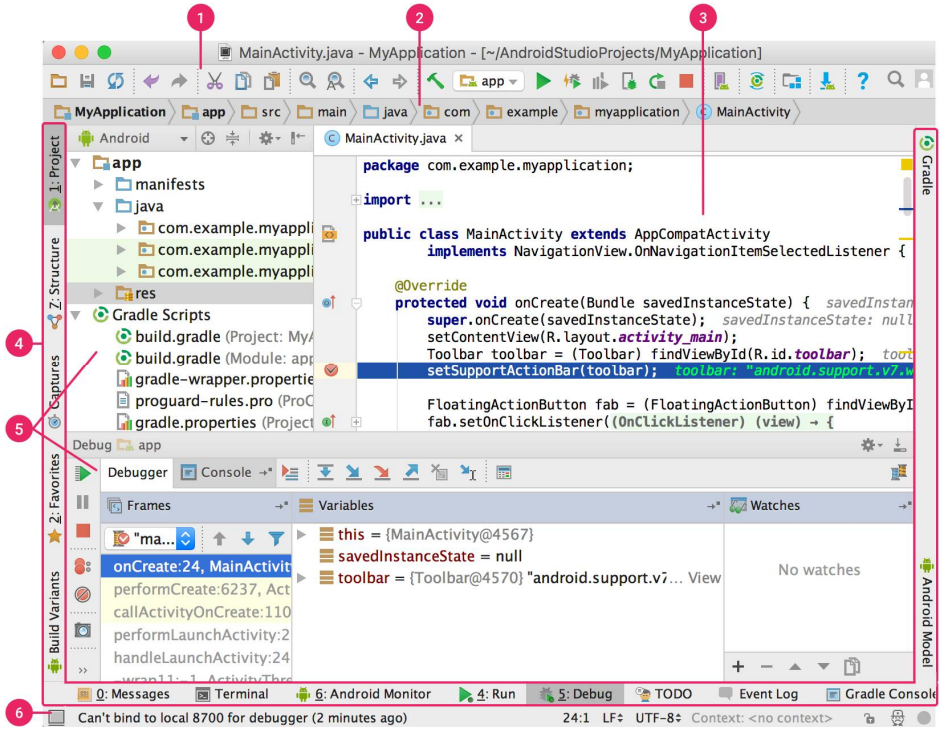


Рисунок 1.2.4. Основные панели в Android Studio.

1) Панель инструментов позволяет выполнить широкий спектр действий, включая выполнение приложения и запуск инструментов Android.
2) Панель навигации помогает перемещаться через проект и открывать файлы для редактирования. Это обеспечивает более компактное представление о структуре, видимой в окне Project.
3) Окно редактора – это то, где создается и изменяется код. В зависимости от текущего типа файла, меняется внешний вид редактора. Например, при просмотре файла расположения, редактор выводит на экран редактора расположения.
4) Панель окна инструментов обтекает за пределами окна IDE и содержит кнопки, которые позволяют разворачивать или сворачивать отдельные окна инструментов.
5) Окна инструментов предоставляют доступ к определенным задачам как управление проектами, поиск, управление версиями, и больше.
6) Строка состояния выводит на экран состояние проекта и самого IDE, а также любые предупреждения или сообщения.

Существует возможность организовать главное окно так, чтобы дать разработчику больше экранного пространства, скрыв или сделав движущимися панели инструментов и окон инструментов. Также можно использовать сочетания клавиш, чтобы получить доступ к большинству функций IDE.

В любое время можно воспользоваться поиском в исходных кодах, базах данных, действиях, элементах пользовательского интерфейса, и т.д., двойным нажатием клавиша Shift или щелчок по лупе в верхнем правом углу окна Android Studio.

Android Studio поддерживает множество систем управления версиями (VCS - version control systems), включая Git, GitHub, CVS, Mercurial, Subversion и Google Cloud Source Repositories.

Android Studio использует Gradle в качестве основы системы сборки с более специфичными для Android возможностями, предоставленными Android плагином Gradle. Эта система сборки работает как интегрированный инструмент из меню Android Studio, и независимо из командной строки. Для разработчика доступны следующие функции системы сборки:

- Настройка, конфигурация и расширение процесса сборки.
- Создание множества APK файлов для приложения, с различными функциями, используя тот же проект и модули.
- Код повторного использования и ресурсы через исходные наборы.

Система сборки может помочь создавать различные версии того же приложения из единственного проекта. Такая функция необходима при наличии бесплатной и платной версии приложения, или если нужно распределить множество APK файлов для различных конфигураций устройства в Google Play.

Android Studio помогает в отладке и улучшении производительности исходного кода приложения, включая встроенную отладку и аналитические инструменты производительности (Рисунок 1.2.5).

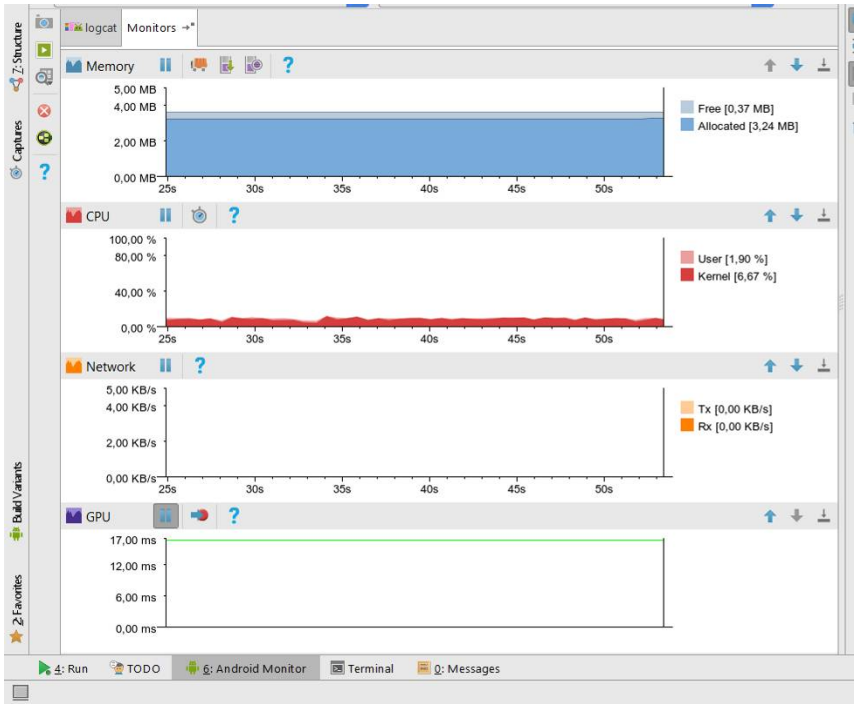


Рисунок 1.2.5. Панель Monitors.

Android Studio строит проект автоматически в процессе его изменения. Во время построения инструментарий Android берет ресурсы, код и файл AndroidManifest.xml (содержащий метаданные приложения) и преобразует их в файл .apk. Полученный файл подписывается отладочным ключом, что позволяет запускать его в эмуляторе. Для распространения файла .apk среди пользователей, необходимо подписать его ключом публикации. [15]

1.2.2 LibGDX

libGDX – кроссплатформенный фреймворк для разработки игр и визуализации, основанный на языке программирования Java с некоторыми компонентами, написанными на C и C++ для повышения производительности определенного кода. В настоящее время поддерживает Windows, Linux, Mac OS X, Android, iOS и HTML5 как целевые платформы. LibGDX распространяется в соответствии с лицензией Apache 2.0 (лицензия на свободное программное обеспечение Apache Software Foundation), что дает право использовать libGDX для любых целей, свободно распространять, изменять, и распространять изменённую копию, за исключением названия.

libGDX позволяет разработчику писать, тестировать и отлаживать код на собственном компьютере и затем переносить его на другие ОС. Для этого фреймворк использует отдельные модули для сборки приложения под каждую платформу, а так же независимый модуль, который содержит основной код приложения. libGDX позволяет написать код однажды и затем развертывать игру или приложение на нескольких платформах без модификации. Так же можно разрабатывать приложение на основном компьютере и получать огромную выгоду быстрой разработки, вместо того, чтобы ждать, когда последние изменения буду внедрены и установлены на устройство и будут скомпилированы в HTML5. Можно использовать все инструменты Java, чтобы быть продуктивным, насколько это возможно.

libGDX позволяет перейти на любой низкий уровень, давая примой доступ к файловой системе, устройствам ввода, аудио устройствам и OpenGL через единый OpenGL ES 2.0 и 3.0 интерфейс.

Наверху таких низкоуровневых возможностей создан мощный набор API, который позволяет решать общие в разработке игр задачи, такие как визуализация спрайтов, теста, построение пользовательских интерфейсов, проигрывание звуковых эффектов и музыки, линейная алгебра и тригонометрические вычисления, разбор JSON и XML, и так далее.

При необходимости, libGDX может перейти от Java к нативному коду, чтобы сосредоточиться на самой лучшей и возможной производительности. Весь этот функционал скрывается за Java API, так что разработчик не должен беспокоится о кросс-компилировании нативного кода на всех платформах. Многие части libGDX знают специфику платформы и разработчику не нужно с ними сталкиваться.

libGDX сосредоточен на том, чтобы являться фрейворком, нежели движком, признавая, что нет универсального решения. libGDX предоставляет мощные абстракции, которые позволяют выбирать, как создавать игру или приложение. [14]

libGDX для своей функциональности использует множество сторонних библиотек:

- Lightweight Java Game Library (LWJGL) - библиотека Java, которая включает межплатформенный доступ к популярным встроенным API, полезным в разработке графики (OpenGL), аудио (OpenAL) и параллельные вычисления (OpenCL) приложения. Этот доступ прямой и высокоэффективный, все же также обернутый в безопасный с точки зрения типов и удобный для пользователя уровень, подходящий для экосистемы Java. [22]
- OpenGL - главная среда для разработки переносимых, интерактивных 2D и 3D графических приложений. Начиная с его введения в 1992, OpenGL

стал отраслью, наиболее широко используемой, и поддерживающей 2D и 3D графические прикладные программные интерфейсы (API), принося тысячи приложений к большому разнообразию компьютерных платформ. OpenGL способствует инновациям и разработке приложений скоростей, включая широкий набор рендеринга, отображения текстур, специальных эффектов и других мощных функций визуализации. [19]

- FreeType - библиотека программного обеспечения в свободном доступе, для визуализации шрифтов. [12]
- OpenAL - межплатформенный 3D аудио API, подходящий для использования с игровыми приложениями и многими другими типами аудио приложений. Библиотека моделирует набор источников звука, перемещающихся в 3D пространство, которые слышит единственный слушатель в пространстве. Основные объекты OpenAL - слушатель, источник и буфер. Может быть большое количество буферов, которые содержат аудиоданные. Каждый буфер может быть присоединен к одному или более источникам, которые представляют точки в 3D пространстве, которые испускают аудио. [23]
- SoundTouch Audio Processing Library - библиотека обработки аудиоданных с открытым исходным кодом для изменения темпа, подачи и скоростей воспроизведения аудиопотоков или аудиофайлов. Библиотека дополнительно поддерживает оценку стабильных уровней ударов в минуту для аудиотреков. [20]
- Vorbis - абсолютно открытая, профессиональная технология аудиокодирования и потоковой передачи без патентов со всеми преимуществами открытого исходного кода. [21]
- mpg123 – это библиотека содержащая работающий в реальном времени MPEG 1.0/2.0/2.5 аудиоплеер / декодер для уровней 1,2 и 3 (обычно уровень 3 MPEG 1.0 иначе MP3), а также допускающие повторное использование библиотеки декодирования и вывода. [17]

- Box2D – это библиотека написанная на C++, с открытым исходным кодом для моделирования твердых тел в 2D. Box2D разработан Erin Catto и имеет zlib лицензию. zlib лицензия не требует подтверждения, что позволяет свободно использовать Box2D в своем продукте. [9]
- Kiss FFT ("Keep It Simple, Stupid." Fast Fourier Transform) - очень небольшая, довольно эффективная, смешанная библиотека FFT основания, которая может использовать или фиксированную точку или типы данных с плавающей точкой. [13]

Кроме того libGDX может быть интегрирован со многими сторонними инструментами, например:

- Spine – мощный и богатый инструмент для создания анимации, в частности и для 2D игр (Рисунок 1.2.6). Минус данного инструмента является то, что он не бесплатный, полная (профессиональная) версия стоит 299$. [24]

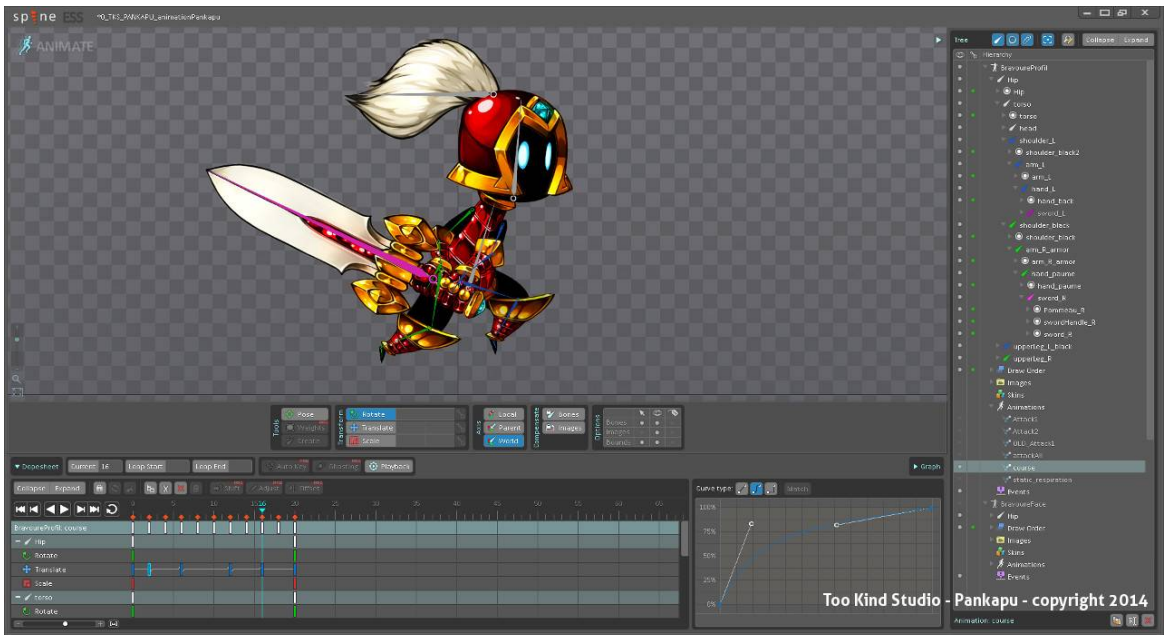


Рисунок 1.2.6. Вид редактора Spine.

- Nextpeer – с помощью данного инструмента можно создать полностью функциональную многопользовательскую игру всего через несколько часов. [18]

- GuardSquare - свободный универсальный оптимизатор и обфускатор (программа, в результате выполнения которого код программы приобретает вид, трудный для анализа) для байт-кода Java. [16]

  В libGDX есть и свои инструменты:

- 2D Particle Editor - мощный инструмент для создания эффектов с помощью частиц (Рисунок 1.2.7).

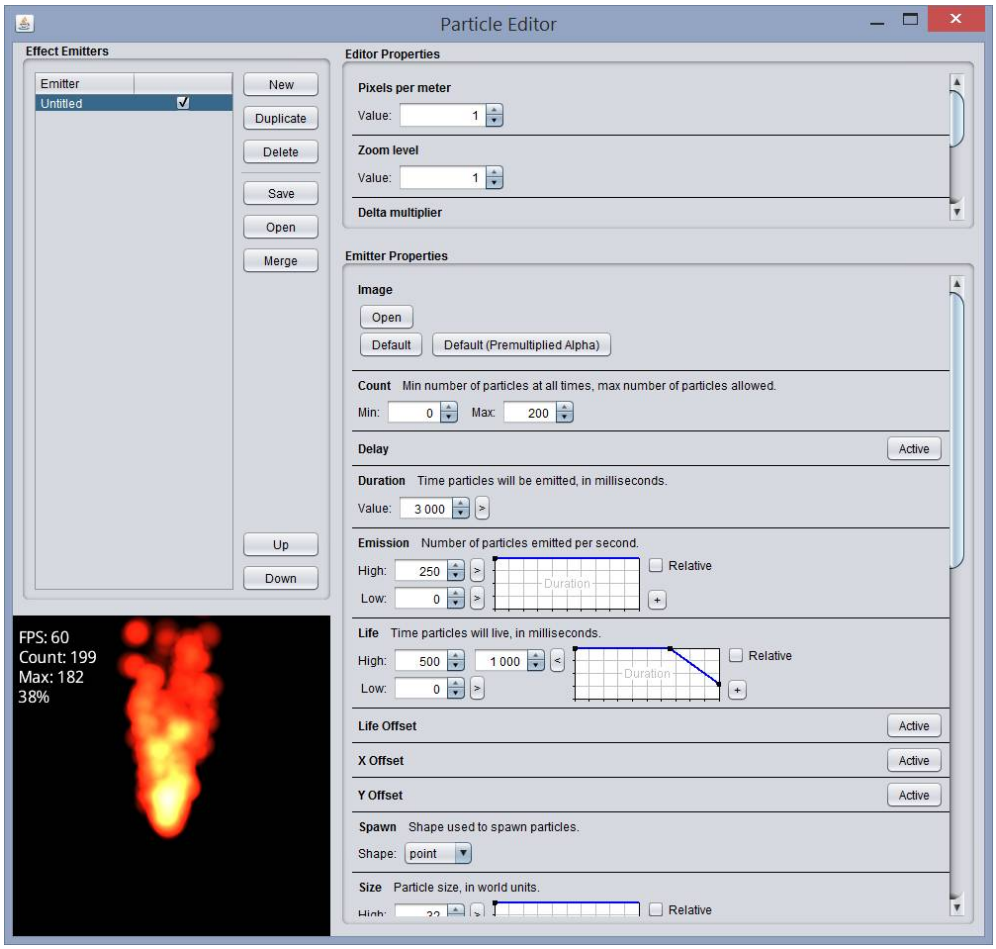


Рисунок 1.2.7. Редактора частиц.

- 3D Particle Editor – тот же редактор для создания эффектов с помощью частиц но уже в 3D.
- Texture packer – инструмент позволяющий упаковывать множество изображений в один «атлас». Это очень удобно если в приложении много изображений (Рисунок 1.2.8).

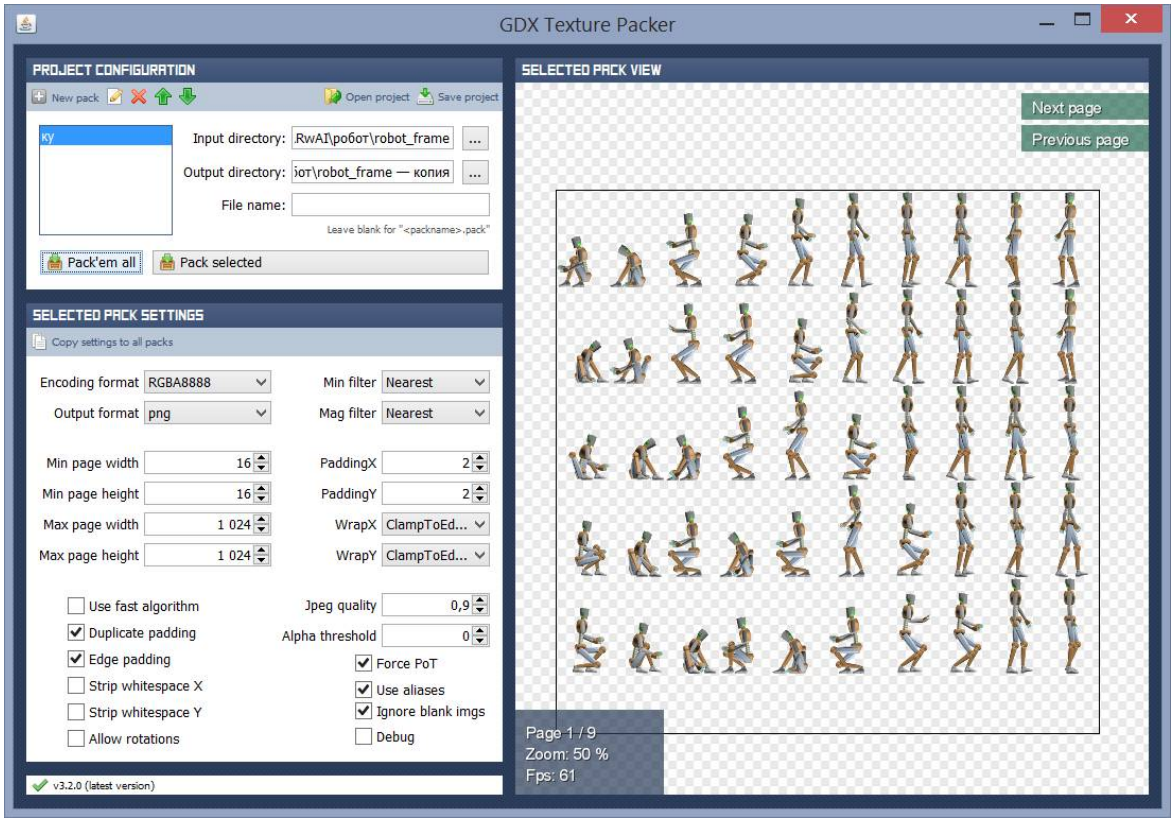


Рисунок 1.2.8. Упаковщик текстур.

- Hiero - инструмент упаковки растрового шрифта (Рисунок 1.2.9).

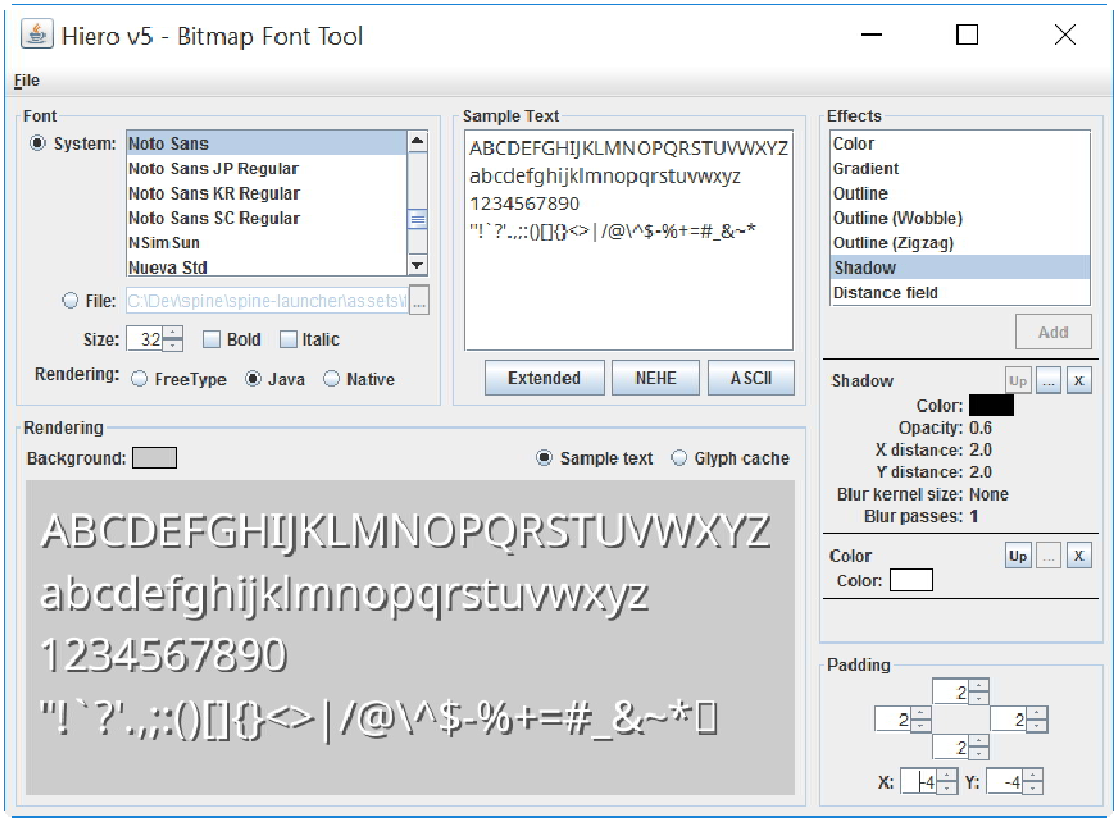


Рисунок 1.2.9. Редактор шрифтов.

Так же с помощью libGDX можно создавать не только 2D игры, но и 3D. На данный момент 3D API в libGDX находится в доработке, но, не смотря на это, с помощью него можно создавать 3D игры. Так же в libGDX есть поддержка шейдеров.

Все выше описанное, позволяет сделать вывод, что Android Studio и libGDX прекрасно подходят для создания нашей 2D игры. Хотелось бы подчеркнуть тот факт, что если создать абсолютно одну и ту же игру с использованием игрового движка Unity, и та же игра, созданная с использованием Android Studio + libGDX, будет иметь различия, по производительности и к объему требуемой памяти. Даже если сравнить два исходных apk файла с пустым проектом (один из них полученный с Unity, а другой с Android Studio + libGDX), то apk файл полученный с Unity будет занимать больше памяти в устройстве, чем apk файл полученный с Android Studio + libGDX (Рисунок 1.2.10).

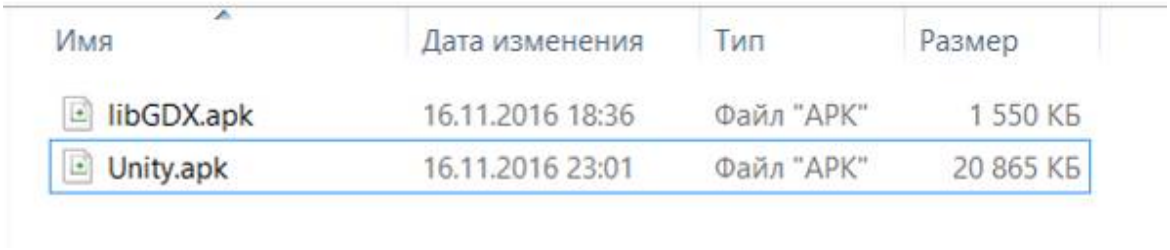
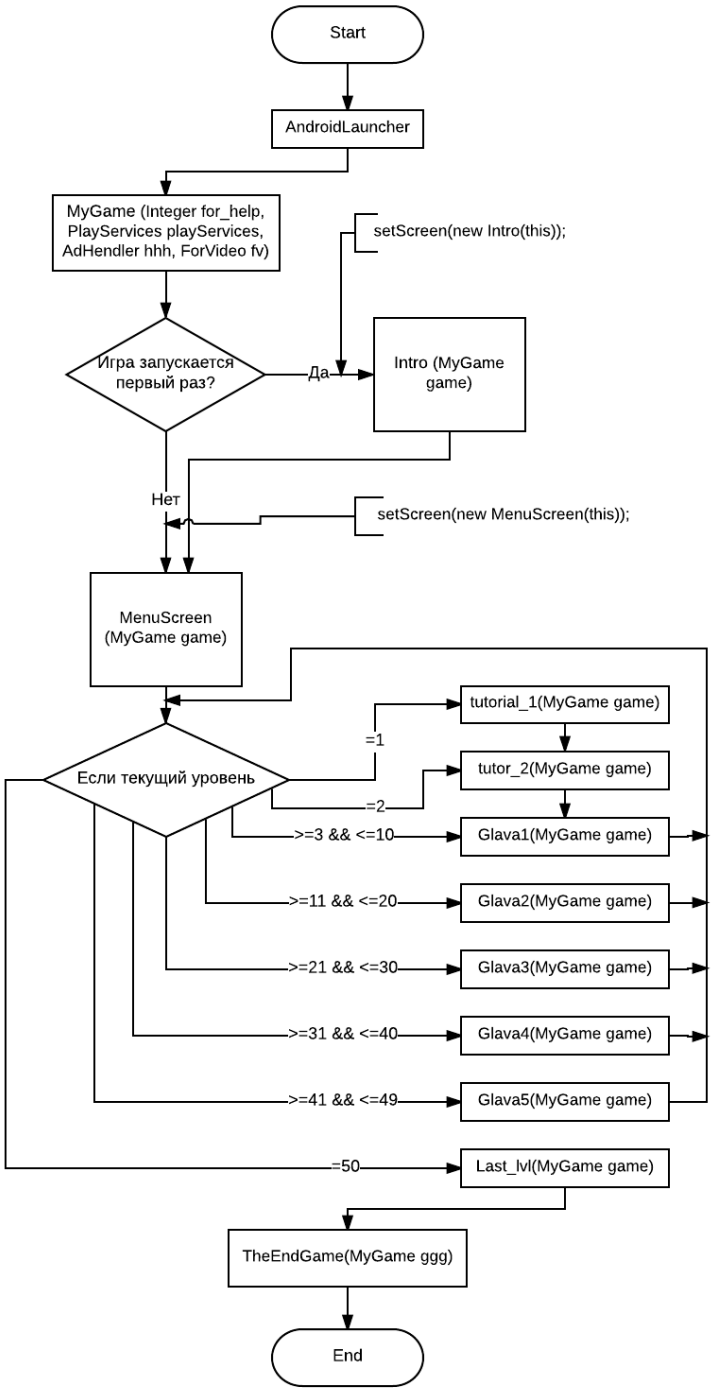


Рисунок 1.2.10. Различие в исходных размерах между "пустыми" проектами Unity и libGDX.

1.3 Технического задание

1. Название системы – Проект АРсИИ: Эксперимент.
2. Назначение – игровое приложение.
3. Функциональные требования – сенсорное управление.
4. Системные и программные требования – ОС Android версии 4.0 или выше, 50 Мб свободного пространства в памяти телефона, 512 Мб ОЗУ.

5. Аппаратные требования – мобильное устройство с сенсорным управлением, разрешением экрана 1280x768 и более.
6. Блок-схема приложения:



Глава 2. Разработка игрового приложения средствами Android Studio и libGDX

2.1 Технология разработки игрового приложения

Многие люди, играющие в игры, даже не представляют себе, сколько трудов и творческих идей вложено в каждую отдельно взятую игру.

Создание игры это продолжительный и трудоёмкий процесс, состоящий из самых разнообразных этапов, включающий в себя как технические, так и творческие моменты. По этой причине, игры создают не отдельные личности, а целые команды разработчиков. Каждый отдельный человек в команде – специалист в своей области знаний.

Первое, что нужно сделать - это определиться с целью. Этапом концепции и определения цели занимается руководитель проекта. На ранних этапах разработки можно представлять в мельчайших деталях готовую игру, так же есть возможность создавать сюжет, стилистику, особенности игры в ходе самой разработки. В этом деле не обязательна излишняя точность, но, как минимум, нужно задать направление развития игрового проекта.

Выбор жанра игры осуществляется в самом начале. Жанр будет основным направлением развития игры. На данный момент в Google play в категории игр, существуют 17 подкатегорий (Рисунок 2.1.1), из них можно выделить самые основные:

- Аркады - этот жанр трудно назвать жанром, аркады являются скорее игровым направлением. Игровое приложение можно называть «аркадной», если она напрямую портирована с игрового автомата или же схожа по концепции с играми для автоматов. Так же данный жанр называют платформером. Для таких игр характерно иметь главного героя, которым управляет игрок, перемещая его по уровню.
- Головоломки – игры, целью которых является решение логических задач, требующих от игрока задействования логики, стратегии и интуиции.

- Казуальные игры – данная категория игр, предназначенная для широкого круга пользователей. Казуальные игры отличаются простыми правилами и не требуют от пользователя особой усидчивости, затрат времени на обучение или каких-либо особых навыков; они относительно дешевы в разработке и при дистрибуции. Многие подобные игры обладают также яркой привлекательной графикой и минимумом текста. Казуальным играм противопоставляют «хардкорные» игры со сложными правилами, рассчитанные на сравнительно узкую аудиторию опытных игроков, готовых уделить много времени освоению игры.
- Обучающие игры – игры тренирующее и обучающее человека в игровом режиме. Может применяться как для обучения, так и для развлечения. Программа делит на части учебный материал, и регулирует последовательность его изучения. Усвоение материала проверяется тестом, предлагаемым в конце каждого этапа обучения.
- Ролевые (RPG - Role-Playing Game) – игры данной категории основаны на элементах игрового процесса традиционных настольным ролевым играм. В ролевой игре игрок управляет одним или несколькими персонажами, каждый из которых описан набором численных характеристик, списком способностей и умений; примерами таких характеристик могут быть показатели силы, ловкости, защиты, уклонения, уровень развития того или иного навыка и т.п. В ходе игры они могут меняться. Одним из характерных элементов игрового процесса является повышение возможностей персонажей за счёт улучшения их параметров и изучения новых способностей.
- Симуляторы – игры имитаторы, задача которого состоит в имитации управления каким-либо процессом, аппаратом или транспортным средством.
- Стратегии – это игры, в которых залогом достижения победы является планирование и стратегическое мышление. Смысл таких игр заключается

в управлении определёнными ресурсами, которые необходимо преобразовать в преимущество над противником при помощи оперативного плана, разрабатываемого с учётом меняющейся обстановки. Обычными ресурсами в военных стратегиях являются войска (отдельные персонажи, подразделения или армии) и позиция, которые следует развивать и использовать для достижения преимущества и победы. В экономических стратегиях акцент ставится на развитие экономической инфраструктуры подконтрольной игроку стороны. Современные стратегические игры, как правило, соединяют в себе как военные, так и экономические признаки.

- Экшен (Action, действие) – в этих играх успех игрока в большой степени зависит от его скорости реакции и способности быстро принимать тактические решения. Действие таких игр развивается очень динамично и требует высокую концентрацию внимания и быстрой реакции на происходящие в игре события.

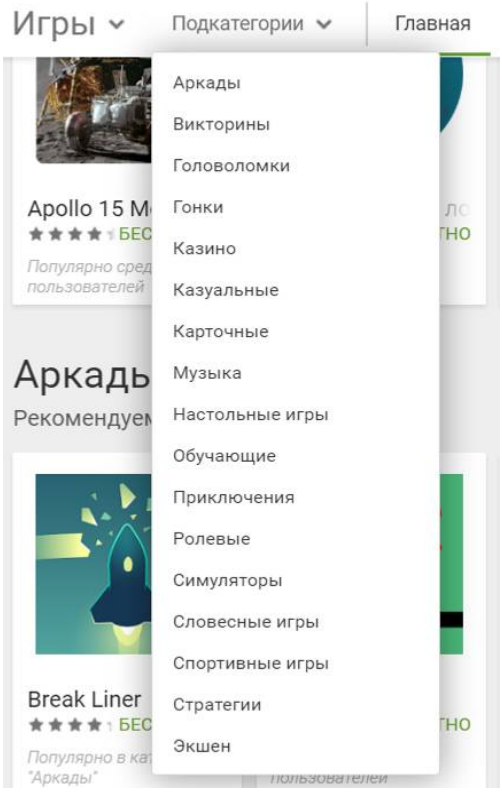


Рисунок 2.1.1. Список подкатегории игр в Google play.

Некоторые элементы жанра принесут игре высокую популярность среди игроков (уничтожение, соревнование, герой, забота). Какие-то – преданных, но придирчивых фанатов (тактика, контроль, уклонение), а какие-то – отсутствие серьезных конкурентов (обучение, логика, путешествие, экономика).

Выбранный жанр можно немного корректировать по ходу работы, но его сущность должна оставаться прежней. Жанр – это своеобразный фундамент всей игры. Если захотите сменить жанр разрабатываемой игры, то целесообразнее начать разработку новой игры заново, чем переделывать то, что уже было наработано.

Разделение компьютерных игр на жанры весьма специфично и не похоже на систему жанров фильмов и книг. Игровые жанры определяют лишь основные действия, которые будут совершать игроки в процессе игры. Время и место этих действий в игре, определяется основной характеристикой называемой – сеттинг.

Сеттинг – это принадлежность игры к определенной сюжетной тематике или к определённому виртуальному миру. В среде компьютерных игр сформировалось несколько наиболее популярных сеттингов: фэнтези, научная фантастика (sci-fi), вторая мировая война, средневековье, стимпанк, постядерный мир, аниме, комиксы.

Создание игры в популярном сеттинге обеспечивает её собственную популярность, игроки чувствуют себя уютно и комфортно в уже знакомом мире. Некоторые игры создаются в своих уникальных сеттингах или в необычных сочетаниях стандартных тем. Такие игры менее популярны, но, тем не менее, они имеют свою аудиторию особых игроков.

После задания цели игрового проекта, нужно выбрать средства (материалы и инструменты) для её достижения. Инструментом и материалом игрового проекта является одна и та же сущность – это программный код. Код как строительный материал – это цифровые изображения, трехмерные модели, звуки и тексты в виде последовательностей единиц и нулей. Код как инструмент – это

команды в строчках программного кода, управляющие игровыми объектами всех перечисленных типов.

Создание игрового материала (наполнения, контента) – это чисто творческая часть процесса. Программный код представляет собой каркас (скелет), на который будут нанизываться результаты всех последующих этапов разработки. Этим этапом занимаются программисты.

Прежде всего, нужно выбрать язык программирования. После этого предстоит тяжелая и кропотливая работа по написанию программного кода, способного оперировать двухмерными или трехмерными объектами в пространстве, привязкой изображений и звуков. Для создания виртуального трехмерного пространства используются сложные геометрические формулы для построения проекции 3D-объектов на плоскость (в памяти компьютера объекты существуют в настоящем трёхмерном пространстве, но для вывода их изображения на плоский двухмерный экран нужно делать пересчеты). По ходу разработки придётся изучить все форматы изображений и аудиофайлов, всевозможные кодеки и кодировки.

Сейчас существуют готовые программные модули (игровые движки), где уже реализованы базовые функции, способные связать воедино графику, звук, объекты и их движения. Таким образом, выбор языка программирования заменяется другой дилеммой – выбором готового игрового движка. Применение игровых движков сводит к минимуму работу программиста.

Самая важная творческая часть любой игры – игровая механика (геймплей). Игровая механика находится не на поверхности, поэтому часто ускользает от взгляда невнимательных ценителей игр. Молодые подростки (основная часть игровой аудитории) в большинстве своём оценивают игры по качеству графики, и не замечают, что красивые игры хоть и популярны, но их популярность длится всего несколько месяцев после релиза. Чем разнообразнее и интереснее игровые возможности, тем дольше игрок остаётся в игре.

Если представить игру в виде живого организма, то игровая механика будет является его нервной системой и головным мозгом. Если представить игру в виде строящегося дома, то игровая механика предстанет как электропроводка, трубопровод и прочие бытовые инженерные коммуникации. Можно представить, что происходит с красивыми и модными, но не продуманными играми с данной точки зрения: можно поселиться в красивом доме, но если в нём не будет освещения, водопровода и канализации, то при первой же возможности человек покинете его в поисках более комфортабельного жилья.

Игровая механика – это свод правил, по которым будет функционировать игра. Основой всей механики являются игровые объекты. Главный герой игры, компьютерные соперники, второстепенные персонажи, бонусы, подвижные объекты, декорации – всё это игровые объекты со своими свойствами и возможными действиями.

Игровая механика определяет, какими клавишами будет управляться главный герой или основной игровой объект, какое действие будет происходить после нажатия той или иной кнопки. Сюда же относится законы поведения игровых объектов (физический движок) и поведение врагов (искусственный интеллект).

Если «управление» отвечает за перемещение подконтрольного игроку персонажа, то физический движок отвечает за те движения, которые происходят без прямого вмешательства игрока. Эти действия имитируют физические законы реального мира (иногда немного искаженные в сторону фантастики). Брошенный мячик отскакивает от пола, опрокинутая бочка скатывается с наклонной поверхности, выстрел мощным оружием отбрасывает стреляющего назад, хрупкий предмет, брошенный с высоты разбивается – всё это примеры действия физического движка.

В готовых игровых движках чаще всего реализованы и физические движки. Разработчику остается присвоить своим уникальным объектам уже готовые физические характеристики: вес, плотность, эластичность, разрушаемость. Если

же создавать свой физический движок, то для этого понадобится талантливый программист, хорошо понимающий принципы объектно-ориентированного программирования (ООП) и немного разбирающийся в классической физике.

Искусственный интеллект (ИИ) отвечает за поведение компьютерных врагов или союзников. Роль ИИ значительно разнится в зависимости от жанра игры. В экшенах действия врагов крайне примитивны; в стратегиях достаточно пары десятков скриптов, чтобы придать сопернику кажущуюся разумность; в стелс-экшенах, слешерах и файтингах необходимо создать уникальную систему поведения для каждого типа врагов, иначе глупые враги сделают игру неинтересной. Серьезная стратегическая игра требует колоссальной работы над ИИ, а в простых казуальных играх и в онлайн-проектах, ориентированных на сражения только между реальными игроками, искусственный интеллект отсуствует.

После создания правил игры в виде игровой механики, нужно создать площадки, где эти правила начнут работать. Созданные игровые объекты расставляются в отдельных виртуальных пространствах – уровнях (локациях). Игры чаще всего содержат множество отдельных уровней, переход между которыми происходит по ходу сюжета. Но в последнее время, благодаря возросшей производительности компьютеров, выпускаются игры с одним большим цельным миром, лишь условно разделяемом на различные локации (GTA, Skyrim).

На каждом отдельном уровне расставляются игровые объекты, стенки, платформы, декорации, фоны. Уровни создаются в играх всех жанров. Если представить игру в виде дома, то построение игровых уровней – это планировка этажей, а количество уровней – этажность здания. Построением уровней занимаются левелдизайнеры.

В идеале левелдизайнеры берутся из числа людей с большим игровым опытом. Это происходит потому, что любой другой человек со стороны, пусть даже и творческий, но очень далёкий от темы игр, не сможет хорошо справиться с этой задачей. Левелдизайнер должен хорошо представлять себе игровой

процесс, и чувствовать, как от перемещения объектов на уровне будет изменяться игровая ситуация.

Довольно часто в комплекте с игрой поставляется редактор уровней, с помощью которого обычные игроки могут самостоятельно создавать себе новые карты и уровни. Разработчики игр поддерживают распространение самодельных карт между игроками и часто выкладывают лучшие работы на своих официальных серверах. Редакторы уровней создаются не только для развлечения игроков и увеличения срока жизни отдельно взятой игры, но и для того, чтобы отыскать среди игровой аудитории наиболее талантливых людей. Таким образом, игровые студии решают свою кадровую проблему.

Именно от дизайна (не оформления, а планировки) уровней зависит важнейшая составляющая игры – геймплей. (Это правило не действует лишь для большинства казуальных игр, файтингов и спортивных игр, где уровни крайне примитивны). Неинтересная и однообразная планировка уровней загубила множество игр с великолепным оформлением, подкреплённым новейшими технологиями.

Далее нужно нарисовать текстуры, изображения для игры. Созданием графической составляющей занимаются художники, геймдизайнеры. При разработке простой 2D-игры, разработчик может самостоятельно создать графическую составляющую. Но в более серьезных и крупных проектах лучше нанять профессиональных художников и дизайнеров.

В начале создаются образы героев, врагов, игровых предметов, задних фонов. Первоначально они рисуются либо на бумаге, либо на компьютере с использованием графического планшета. Для небольших игровых студий этот этап не обязателен, но он просто необходим в больших компаниях, для наглядности изображений, чтоб объяснить всем дизайнерам, что должно в итоге получиться. На основе артов дизайнеры создают либо двухмерные спрайты из пикселей, либо трёхмерные модели из полигонов.

Для игровых объектов, которые будут передвигаться в ходе игры, создаются анимации. Особенно для героев и врагов, количество анимации которых иногда превышает целую сотню различных движений.

Визуальные спецэффекты – это, по сути своей, те же анимации, только вместо перемещения объектов в них используются перемещения частиц и светофильтров. Лучи света в разные стороны при взятии бонусов, огонь на горящем здании, дымовая завеса после взрыва гранаты, лазерные лучи из дула винтовок, наложение фильтров размытия при нахождении под водой и фильтров затемнения в плохо освещённых местах – всё это спецэффекты. Без подобных эффектов игра будет казаться пресной и слишком обыденной. Использование спецэффектов добавляет игре яркости, сочности и экспрессивности.

Оформить нужно не только игровые уровни, но и систему, объединяющую их в единое целое – игровое меню (строчки, кнопки, страницы настроек). Начальное меню – это визитная карточка игры, и выглядеть она должна идеально. На игровом экране так же есть множество элементов, к которым можно применить оформление – количество жизней, миникарта, меню быстрого выбора действий, инвентарь героя, списки заданий, экраны диалогов. На английском языке всё это называют одним сокращением – GUI (Graphical User Interface - графический пользовательский интерфейс). Оформлением интерфейса и меню занимаются художники, программисты и верстальщики html-страниц.

Для привлечения игрока к своему проекту, для разработчика это очень сложная задача, но ещё сложнее сделать так, чтобы игрок прошел игру до конца. Любое разочарование или трудно проходимый уровень, может мгновенно оттолкнуть игрока от дальнейшей игры. Только грамотно поданный качественный сюжет может заставить игрока пройти всю игру, а значит - дослушать интерактивную историю до конца.

На заре своего существования компьютерные игры обходились без сюжета, затягивая игроков лишь своим игровым процессом. Но в настоящее время

даже в самой простой казуальной игре присутствует сюжет, не говоря уже о крупных игровых проектах AAA-класса.

Если сюжет в игре присутствует только для «галочки», то это не даёт никакого положительного эффекта. Сюжет приносит пользу лишь в том случае, если он может зацепить чувства игрока. Для этого нужна уникальность, интересность и правдоподобность сюжета; каждый персонаж должен иметь свою неповторимую личность, и совершать поступки согласно ей; действующих лиц и событий должно быть не больше, чем может воспринять один человек, иначе сюжет станет сложным и не понятным; события должны происходить логично (иногда в сюжет вносят элементы загадочности для поддержания сюжетной интриги, но при этом скрывается от игрока логичность).

Для создания действий по сюжету, в игровом движке существует редактор скриптовых сцен. Скрипт представляет собой следующее: игрок заходит в определённое место, или совершает нужное действие, или выполняются ещё какие-то необходимые условия, и после этого начинают выполняться действия, запрограммированные на этот случай. Например, в военном 3D шутере игрок поднимается на возвышенность, подходит к установленному пулемёту (условие выполнено), через 10-15 секунд после этого внизу, неожиданно начинается вражеская массированная атака, и игроку есть на кого использовать пулемёт (произошли события).

С помощью скриптовых событий можно вносить разнообразие в игровой процесс. Единственный минус такого способа – у игрока уменьшается свобода действий. Всё происходит по воле скриптов, и мало зависит от действия игрока. Продумыванием скриптов занимаются сценаристы, а их реализацией – программисты.

В старых классических играх сюжет существует обособленно от игрового процесса. Например, при загрузке или окончании уровней игрока знакомят с сюжетной историей, рассказывают об отношениях между героями и врагами, объясняют, что и для чего нужно сделать на уровне. В процессе самой игры ни-

чего из вышесказанного не имеет никакого значения, и игрок может смело пропускать все эти тексты. Чаще всего так и происходит - тексты остаются не прочитанными, потому что данные тесты не несут смысла.

В процессе игры, в безопасных местах, или с остановкой игрового времени, игрок может начать диалог при взаимодействии с каким либо игровым объектом, для того чтобы игрок мог сосредоточиться только на тексте. Повествования игроку приходится выслушивать, так как игра на это время замирает, но не останавливается совсем. Иногда в диалогах нужно выбирать вариант ответа. Выбор варианта придаёт прослушиванию текста интерактивность и практический смысл – правильно выбранный ответ может принести дополнительный бонус, облегчить дальнейшую игру или сохранить выбранный характер героя в ролевых играх.

При создании игры тексты повествований и диалоги находятся в отдельных файлах, подгружаемых во время игры. Отделение художественного текста от технических кодов поможет в будущем, при создании локализованной версии игры на других языках мира. Написанием текстов и диалогов занимаются сценаристы и писатели.

Между уровнями игры или в определённых контрольных точках уровней можно вместо сухого текста и озвучки показывать игрокам видеовставки (катсцены). Такие заставки можно создавать как с помощью отдельных видеофайлов, так и с помощью игрового движка.

Видеофайлы позволяют передавать игроку картинку любого качества и сложности, но при создании дистрибутива игры они занимают большое количество дискового пространства. Заставки, создаваемые на движке игры, по своему качеству уже почти не уступают заранее созданными видеозаписям, но для их хорошего просмотра у игрока должен быть достаточно мощный компьютер, что не всегда бывает. Созданием видеовставок занимаются художники, аниматоры, 3D-модельеры, сценаристы, режиссеры.

После графического оформления, в игру добавляют звуковые эффекты. Для любого игрового движения добавляется соответствующий звук. Это могут быть удары меча, нанесение рукопашного удара, звуки движения автомобиля, получение бонуса, обнаружения героя врагом. Можно обойтись всего несколькими базовыми звуками. Например, в большинстве 3D-Action игр пренебрегают добавлением звуков шагов главного героя и врагов. В результате – при виде от первого лица видно, что герой передвигается в пространстве, но кажется, что он не идёт, а плавно скользит вперёд. А враги без озвучки шагов могут совершенно бесшумно подбежать к герою сзади и сильно попортить нервы игрокам. Хорошие звуковые эффекты не только заполняют тишину, но и являют собой продолжение графического стиля игры.

Чаще всего в качестве звуковых эффектов используются реальные звуки, записанные в цифровом виде. Кроме звуков для полноценной игры нужна и музыка (саундтрек). Используется в качестве звукового фона для происходящего на экране. Музыка так же является одним из стилистических элементов игры, и сильнее всего влияет на настроение игрока. Готовую музыку нужно выбирать по подходящему темпу и настроению. Существуют множество как платных, так и бесплатных коллекции игровых фоновых композиций, которые можно использовать. Или можно заказать композиторам написать новую музыку специально под игру.

Третьей звуковой оставляющей игры является озвучка игровых диалогов и монологов. Эта составляющая очень дорога, но её наличие в игре не обязательно. В некоторых играх диалогов и текстов почти нет, а там, где есть, их можно оставить не озвученными в виде текстовых субтитров. Небольшие игры обходятся совсем без озвучки, а в больших проектах для озвучивания даже приглашают профессиональных актеров.

Наличие в игре музыки и полноценной озвучки значительно увеличивает объем готовой игры, занимаемый в памяти устройства. По возможности лучше добавлять озвучку в игру. Это повысит вовлеченность игрока в сюжет и расска-

зываемую историю, так как большинство игроков игнорируют и не читают обычные не озвученные тексты.

Процесс разработки большой игры построен таким образом, что различными её элементами занимаются различные специалисты. На начальном этапе игра представляет собой разрозненный набор творческих наработок в различных областях искусства: изображения, звуки, 3D-модели, архитектура, тексты, сценки, видеовставки, оформление. После создания всего необходимого игрового контента, с помощью программных средств, все объекты соединяются в единую сложную систему.

При построении игры на игровом движке объединение объектов происходит постепенно с самого начала процесса. Пока игра не собрана до конца, её называют альфа версией. В этот момент уже можно заниматься тестированием отдельных уровней, скриптов и прочих механизмов. На этом этапе уже технически возможно выпустить демонстрационную версию (демо-версия) или видеоролик с игровым процессом, для привлечения игроков к проекту.

Когда игра полностью собрана, остаётся лишь устранить получившиеся ошибки (bugs, баги). Они появляются в любом случае, так как игра – это система со сложной структурой. Сами элементы игры наглядны и просты, но связи между ними настолько сложны и витиеваты, что процесс отладки и устранения ошибок может занимать до 40% всего времени разработки проекта. Полностью собранная, но ещё не проверенная на ошибки игра называется бета версией.

Поиском ошибок в игре занимаются тестировщики. Очень часто в качестве тестеров привлекаются группы обычных игроков, и это служит началом их карьеры в игровой индустрии. Проще всего эта проблема решается в онлайн-играх – разработчики организовывают открытые бета-тесты, в которых участвуют все желающие игроки.

Созданием игры и всеми творческими вопросами занимается студия разработчиков, а все прочие вопросы (кредиты, финансы, договора, защита прав,

рекламные акции, локализации, продажи) обычно перекладываются на плечи другой организации – игрового издателя.

Отношения между разработчиками и издателями могут быть самыми разными: договор о сотрудничестве на равных правах; все права, финансовые риски и возможная прибыль принадлежит издательству, а разработчики лишь получают свой небольшой процент от прибыли; все права у разработчиков, издательство – отдельная фирма, временно нанятая разработчиками; разработчики и издательство являются разными подразделениями одной большой корпорации.

Прежде чем продать игру конечному пользователю, издателям для начала нужно сообщить о существовании этой игры. Игру могут купить ничего о ней не зная, просто выбрав в магазине наугад, но шанс, что таким образом выберут именно данную игру, крайне низок. Намного выгоднее распространить информацию об игре по всем возможным каналам. Для этого используют либо рекламу в магазине компьютерных дисков, либо рекламу на интернет ресурсах.

Игровая индустрия не похожа на рынок обычных товаров, у неё есть свои особенности. Информация среди активной игровой аудитории разносится с молниеносной скоростью и охватывает всех вокруг. С такой особенностью самой эффективной рекламой игры является её высокое качество. Если данная игра будет интересна и увлекательна, то о ней совершенно бесплатно напишут игровые журналы и информационные интернет порталы, игроки начнут обсуждать игру и разносить информацию всё дальше, а после прохождения первой игры игроки будут ждать дополнений и продолжений этого проекта. Таким образом, не потратив ни копейки на рекламу, но заслужив уважение у игроков, разработчик может обеспечить успех и текущей игре, и всем последующим дополнениям.

После выпуска игры на родном языке, аудитория потенциальных игроков будет состоять всего из нескольких стран, а прибыль будет минимальной. Выпускать игру на английском языке гораздо выгоднее – её текст будет понимать

большая часть игроков по всему свету, эти игроки будут гораздо более платежеспособными, а значит и прибыль будет на порядок больше.

В идеале, нужно выпускать игру сразу на нескольких самых популярных в мире языках (английском, немецком, французском, испанском, китайском, японском), но для этого нужно иметь целый штат переводчиков и локализаторов. Причём, желательно, чтобы переводчики были носителями языка. На свой родной язык они смогут перенести максимум смысла оригинального текста.

Значительно упростить процесс локализации поможет отделение художественного текста от технической части игры. Для этого тексты, субтитры и аудиофайлы озвучки размещаются в отдельных легкодоступных файлах стандартных типов. А в программном коде игры оставляют лишь ссылки, чтобы текст подгружался в игру из этих файлов. Наиболее популярные игры переводятся игроками-энтузиастами самостоятельно, без участия разработчиков.

После того как игра готова и игроки в ожидании её релиза, остается решить как доставить игру конечным пользователям. Классический способ (выпуск большого тиража компьютерных дисков, и продажа их через розничные магазины) всё ещё актуален, но подходит лишь для крупных компаний, и для игр, имеющих хоть изначальную популярность.

Для небольших групп разработчиков подходит распространение игры через системы цифровой дистрибьюции (крупные онлайн-магазины). Такой вариант обеспечивает для игры уже готовую аудиторию покупателей, которая сформировалась вокруг сервиса. Самый известный пример – сервис Steam. Благодаря огромной аудитории игроков, пользующихся Steam, почти каждая игра, вышедшая в этом онлайн магазине, сразу же приобретает мировую известность.

Так же можно создать собственный интернет-магазин с одним единственным товаром – созданной игрой. Но в таком случае придётся рекламировать не только игру, но еще и интернет-адрес магазина, и завоёвывать аудиторию самостоятельно.

Создание игры и её продажа – это не конец жизненного цикла игрового проекта. Когда игра уже находится у конечных пользователей, игрокам ещё может понадобиться ваша помощь. У крупных компаний существуют даже целые отделы технической поддержки, занимающиеся такими вопросами.

Предшествующий бета-тест устранивший из игры самые очевидные ошибки, не означает, что ошибок совсем в игре не осталось. Очень часто бывает, что массовое использование игры вскрывает более мелкие и незаметные ошибки, которые не смогли обнаружить небольшие группы бета-тестеров. Это могут быть проблемы из-за несовместимости с мало популярными марками оборудования, или ошибки из-за неестественного использования игровых возможностей.

Все это способствует тому, что часто приходится вносить исправления ошибок в уже готовую игру. Такие исправления называются патчами, и этот термин очень распространен в игровой индустрии. Мало кому удаётся сразу же выпускать идеальные игры, чаще всего игры доводятся до идеала уже после своего официального релиза.

Если за игру разработчик получил от игроков деньги за каждую проданную копию, то за выпуск патчей разработчик не получаете ничего. С рыночной точки зрения, выпуск патчей – убыточное и бесполезное действие, которое не обязательно выполнять. Но в реалиях игровой индустрии если разработчик не поддерживаете свой продукт до конца, то он получит дурную репутацию у игроков, и теряете возможную будущую прибыль. Несмотря на свою бесплатность, выпуск патчей - это очень полезное и нужное дело.

Интересная и увлекательная игра без серьезных изъянов и ошибок даёт разработчику зелёный свет для дальнейшей творческой деятельности. Если игровое приложение обрела успешность, то к уже готовой игре можно готовить дополнение или полноценную вторую часть, а начатый сюжет можно развить дальше, превратив его в целую эпопею или даже в полноценную игровую вселенную. Снова создается план разработки игры для создания нового произведе-

ние искусства по тем же самым этапам, но теперь уже с накопленным опытом и приобретенными навыками. [8]

2.2 Разработка игрового приложения

Игровое приложение было создано на основе технологии описанной в разделе 2.1, с использованием IDE Android Studio и фреймворка libGDX. Игра создана с использованием 2D графики, относится как жанру головоломок и приключений.

Название игры – «Проект АРсИИ: Эксперимент» (АРсИИ – автономный робот с искусственным интеллектом). Главным героем является робот, которым управляет игрок, перемещая его по лабиринту. Цель игры – это пройти лабиринты. Лабиринты генерируются по алгоритму Эллера, в лабиринтах случайным образом встречаются стены, которые можно убрать, тем самым сократив себе путь до выхода или же наоборот сделать путь более длинным. Чтобы убрать такую стену, нужно решить математический пример или отгадать число. Примеры и числа генерируются случайным образом.

Главный персонаж был создан в программе 3ds max 2015 как 3D модель, затем анимирована по кадрам. Каждый кадр был сохранен в виде рисунка в формате PNG. Все рисунки были объединены в один большой файл с помощью программы Texture packer (Рисунок 2.2.1).

Рисунок 2.2.1. Кадры анимации (спрайт-лист) главного персонажа.

В графическом редакторе Photoshop CS6 были нарисованы картинки для лабиринтов размером 200x200 пикселей и затем объединены в один общий файл с помощью программы Texture packer (Рисунок 2.2.2).

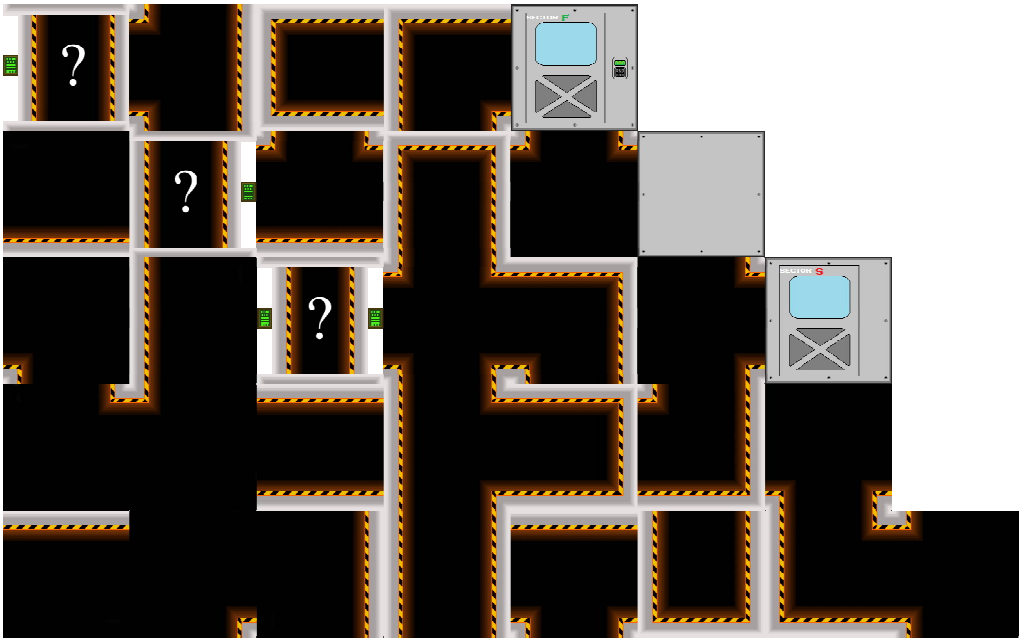


Рисунок 2.2.2. Тайл-сет для лабиринта.

Каждая картинка была нарисована таким образом, чтоб их можно было состыковывать друг с другом, получая при этом единую картину (Рисунок 2.2.3). Такой метод называется тайловая или плиточная графика. Тайлы — небольшие изображения одинаковых размеров, которые и служат фрагментами большой картины. Обычно тайлов на один «мир» делают порядка нескольких сотен. Матрица клеток же составляется из номеров тайлов. Таким образом можно строить огромные двухмерные пространства, расходующие совсем немного памяти.

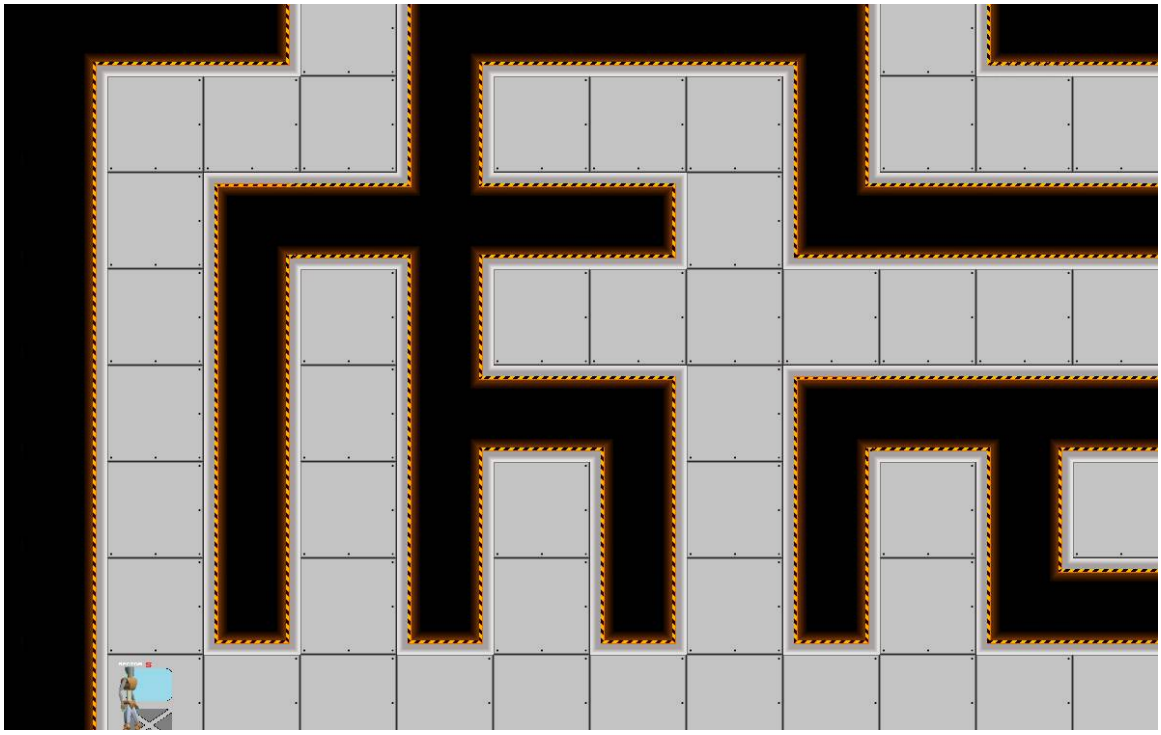


Рисунок 2.2.3. Вид лабиринта построенного из тайл-сета.

Далее в 3ds max 2015 была смоделирована сцена для заднего фона меню игры. В Photoshop CS6 были нарисованы необходимые элементы для интерфейса меню игры, которые так же были упакованы в единый файл с помощью Texture packer (Рисунок 2.2.4).

Рисунок 2.2.4. Набор текстур для главного меню.

Так же в Photoshop CS6 были нарисованы элементы пользовательского интерфейса и упакованы в один файл с использованием Texture packer. Данные элементы необходимы для управления игровым персонажем с помощью сенсорного экрана (Рисунок 2.2.5).

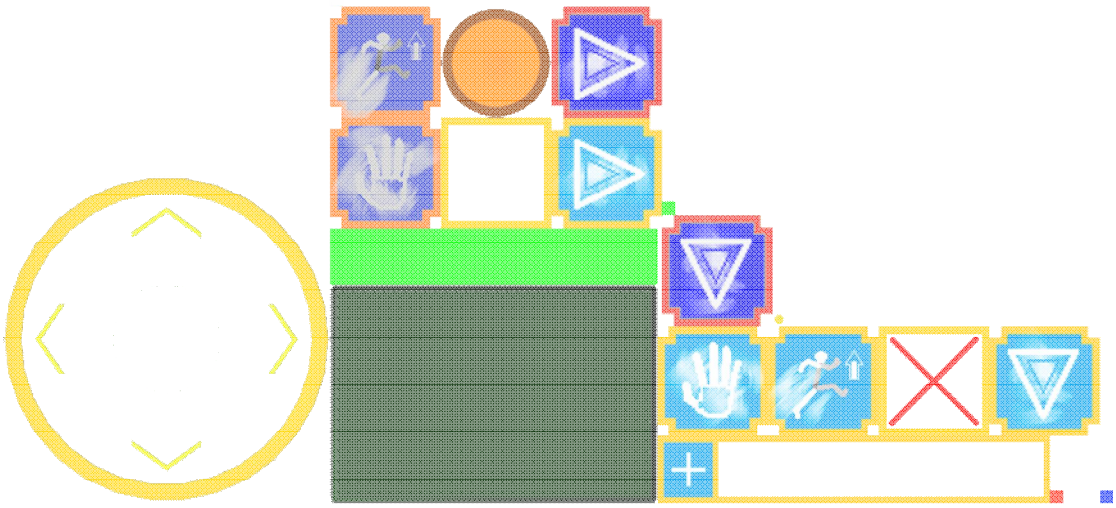


Рисунок 2.2.5. Текстуры для пользовательского интерфейса.

В программе Hiero было создано несколько шрифтов (Рисунок 2.2.6). Шрифт для главного меню и игрового процесса были сделаны на основе шрифта Time New Roman, буквы имеют градиентную заливку. Шрифт для математических примеров сделан на основе шрифта AA Futured, данный шрифт имеется в свободном доступе в сети интернет и его можно использовать бесплатно.

Рисунок 2.2.6. Наборы шрифтов.

После создания всех необходимых графических элементов, был создан пустой проект в libGDX (Рисунок 2.2.7) и импортирован в Android Studio. Проект изначально содержит 3 папки: android, core и desktop. В папке android содержатся все необходимые файлы для платформы Android, в папке core содержится ядро игрового приложения, в папке desktop содержатся файлы для запуска игры на ПК. При разработке удобно производить откладку игры на ПК, а затем устанавливать и проверять либо на реальном устройстве под управлением ОС Android или на эмуляторе. [4]

Рисунок 2.2.7. Главное окно фреймворка libGDX для создания нового проекта.

Все необходимые ресурсы (графика, звуки, шрифты и т.д.) хронятся в папке android/assets (Рисунок 2.2.8). Весь код связанный только с игрой,

находится в папке core. В данной папке происходят все основные действия при разработке игры. Разработчик пишет код один раз, в одном месте, затем на основе папки с проектом core происходит сборка приложения для любой платформы, в данном случае возможно произвести сборку и запуск игрового приложения на ПК и на устройствах, или эмуляторах, под управлением ОС Android.[5]

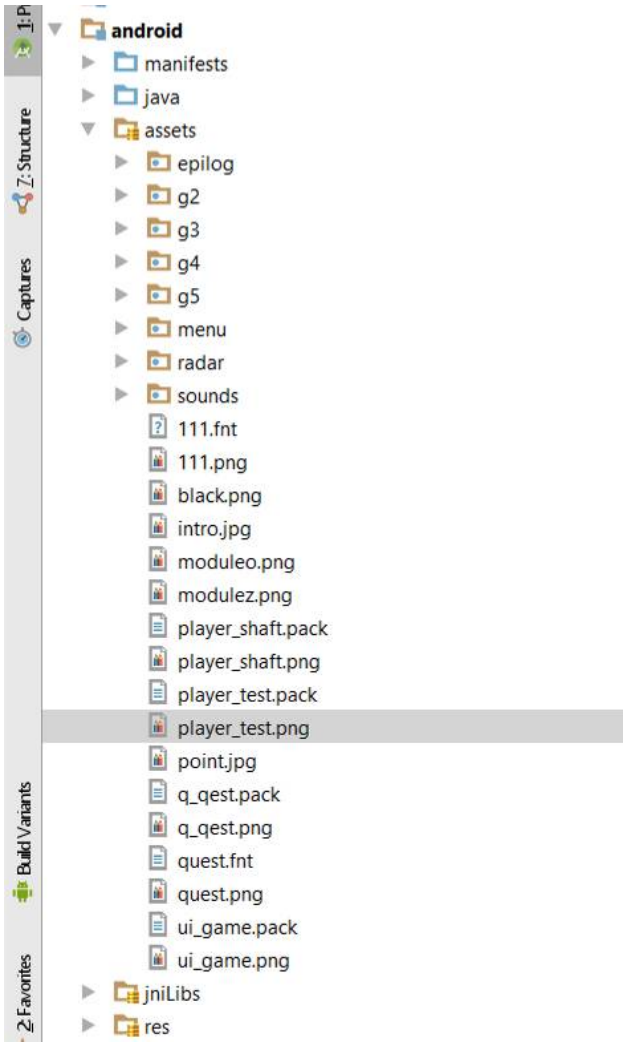


Рисунок 2.2.8. Содержимое папки assets.

При создании данного игрового приложения был использован объектно-ориентированный подход. Игра состоит из 23 классов и 3 интерфейсов (Рисунок 2.2.9).

Рисунок 2.2.9. Список классов и интерфейсов данного проекта.

15 классов являются «экранами» (меню, игровой процесс и т.д.), 6 классов используется для создания экземпляров игровых объектов (робот, лабиринт, математический пример), класс GestureHandler обрабатывает жесты на сенсорных экранах (зум, перемещение какого либо объекта). [6]

На блок-схеме (Рисунок 2.2.10) представлено строение данного игрового приложения и взаимодействие основных классов между собой.

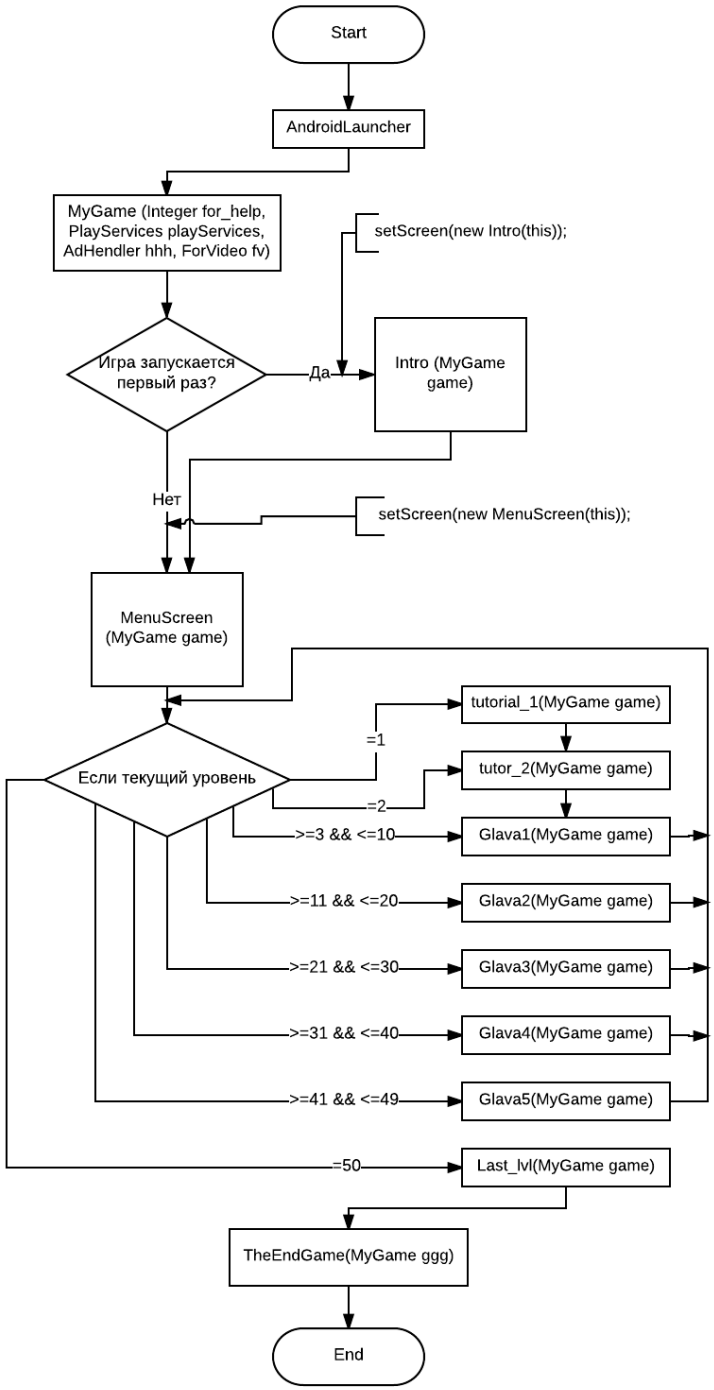


Рисунок 2.2.10. Блок-схема.

MyGame является основным классом, в котором происходит: инициализация основных переменных (максимальное количество пройденных уровней, текущий уровень), объявление методов для сохранения игры и переменных, проверка наличия подключения к сети интернет (если есть доступ к сети интер-

нет то происходит вход в аккаунт игрового сервиса Google play), проверка на первый запуск.

Если игра запущена первый раз, на данном устройстве, то вызывается метод, который «устанавливает» на дисплей устройства экземпляр класса ScreenAdapter (адаптер экрана) указанный в параметре данного метода. В качестве параметра создается экземпляр класса Intro, который наследован от класса ScreenAdapter, ему передается в качестве параметра текущий созданный экземпляр класса MyGame. Если игра запущена не впервые, то так же вызывается метод для «установки» экрана и в качестве параметра передается новый экземпляр класса MenuScreen, который так же унаследован от класса ScreenAdapter, в качестве параметра так же передается текущий созданный экземпляр класса MyGame.

Передача экземпляра класса MyGame нужна для доступа к методам, которые осуществляют сохранение переменных (например, максимальный уровень, до которого дошел игрок). Так же это дает возможность управлять игровыми сервисами Google play и рекламой в игре.

После перехода на экран меню, на экране рисуется задний фон и пользовательский интерфейс (Рисунок 2.2.11).

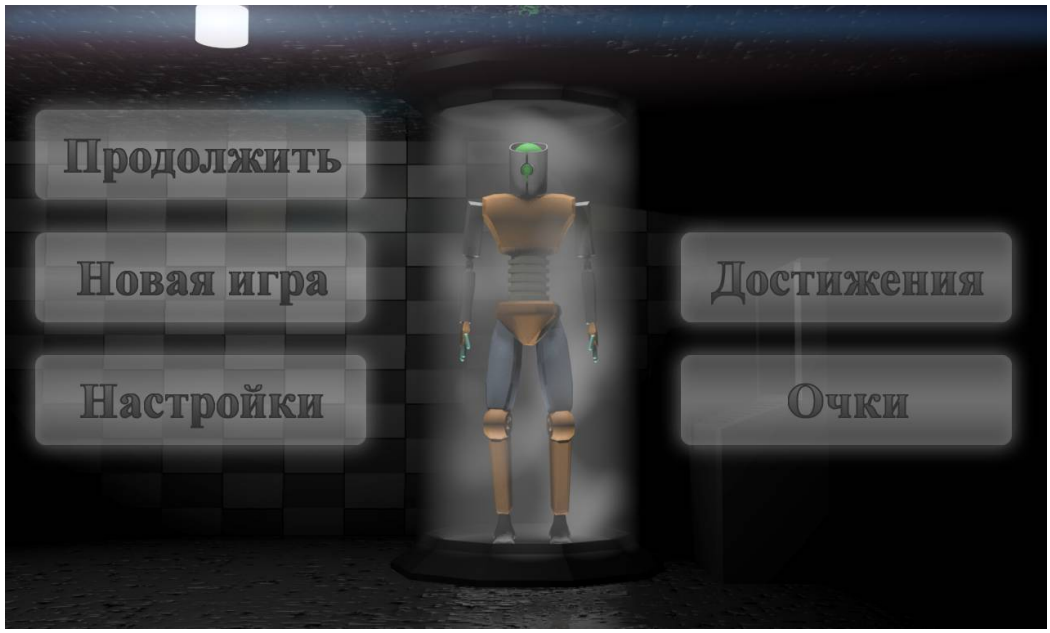


Рисунок 2.2.11. Экран главного меню.

В классе MenuScreen объявляются необходимые переменные для меню:

- Текстуры
- Кнопки для: начал игры, меню настроек, возврата в предыдущее меню, просмотра достижений, просмотра очков (рейтинга игроков по очкам), выбора уровня.
- Слайдеры для настройки звуков (фоновой музыки и звуковых эффектов).
- Шрифты

В конструкторе класса инициализируются объявленные переменные и создаются экземпляры классов.

Так же в конструкторе устанавливаются позиции для каждой кнопки, добавляются в сцену и скрываются те, которые на данный момент не должны быть в главном меню.

Далее инициализируется текущее состояние меню, устанавливаются значения переменных, отвечающие за плавное затенения меню (данный эффект придает ощущения плавности переходов в меню) и для сцены устанавливается обработка ввода данных с устройств.

В преопределенном методе render(float delta) происходит очистка экрана и в данном методе находится логика работы меню игры. Механизм работы меню зависит от двух переменных mstate и pstate, которые являются по типу перечислениями MenuState.

```
    enum MenuState{
        MainMenu, NewGame, SelectLivel, Options, BackOpt, BackN, Resume, Se-
lectN, IamStand, MainSelect, BackS}
```

В зависимости от значения переменной mstate на экране отображаются те или иные элементы меню. Переменная pstate является вспомогательной переменной, используется для скрытия (отображения) кнопок и плавного перехода. Значения mstate и pstate меняется в зависимости от выбранного пункта меню.

Функция is_select_level() вызывается, если была нажата кнопка для выбора уровня. В данной функции определяется, какой уровень был выбран. Далее

переменной curentLvl (текущий уровень) присваивается значение выбранного уровня и функция возвращает значение true.

```
    public boolean is_select_level(){
        for (int i =0;i<select_level.length;i++){
            if(select_level[i].isPressed() &&
TimeUtils.timeSinceMillis(timep)>presstime) {
                if (game_forScreen.effectVol>0f)
btn_b.play(game_forScreen.effectVol);
                game_forScreen.curentLvl=game_forScreen.curentLvl+(i+1);
                timep = TimeUtils.millis();
                return true;
            }
        }
        return false;
    }
```

Игра поделена на 5 глав, в каждой главе по 10 уровней. Всего в игре 50 уровней. После нажатия на кнопку «Новая игра» вызывается метод for_select_glava(). В данном методе определяется количество доступных глав, которые зависит от максимального пройденного уровня. После чего в меню появляются кнопки для выбора уровня.

```
    public void for_select_glava(){
        int sk=game_forScreen.max_lvl_1;
        if(sk>10) G2.setVisible(true);
        if(sk>20) G3.setVisible(true);
        if(sk>30) G4.setVisible(true);
        if(sk>40) G5.setVisible(true);
        G1.setChecked(false);
        G2.setChecked(false);
        G3.setChecked(false);
        G4.setChecked(false);
        G5.setChecked(false);
    }
```

После выбора уровня, вызывается метод for_select_resume() в котором, в зависимости от значения переменной curentLvl, вызывается метод в экземпляре класса MyGame для установки экрана. Если curentLvl = 1, то на экране отображается первый обучающий уровень. Если curentLvl = 2, то на экране отображается второй обучающий уровень и т.д.

```
    public void for_select_resume(){

        if(game_forScreen.curentLvl==1)
game_forScreen.setScreen(new Sz(game_forScreen));
        else if(game_forScreen.curentLvl==2)
game_forScreen.setScreen(new tutor_2(game_forScreen));
        else if (game_forScreen.curentLvl>2 && game_forScreen.curentLvl<=10)
game_forScreen.setScreen(new Glava1(game_forScreen));
```

```
        else if (game_forScreen.curentLvl>10 && game_forScreen.curentLvl<=13)
game_forScreen.setScreen(new Glava1(game_forScreen));
        else if (game_forScreen.curentLvl==14)
game_forScreen.setScreen(new BadEnd(game_forScreen));
        else if (game_forScreen.curentLvl==11)
game_forScreen.setScreen(new Sz(game_forScreen));
        else if (game_forScreen.curentLvl>10 && game_forScreen.curentLvl<=20)
game_forScreen.setScreen(new Glava2(game_forScreen));
        else if (game_forScreen.curentLvl>20 && game_forScreen.curentLvl<=30)
game_forScreen.setScreen(new Glava3(game_forScreen));
        else if (game_forScreen.curentLvl>30 && game_forScreen.curentLvl<=40)
game_forScreen.setScreen(new Glava4(game_forScreen));
        else if (game_forScreen.curentLvl==47)
game_forScreen.setScreen(new Sz(game_forScreen));
        else if (game_forScreen.curentLvl>40 && game_forScreen.curentLvl<=49)
game_forScreen.setScreen(new Glava5(game_forScreen));
        else if (game_forScreen.curentLvl==50)
game_forScreen.setScreen(new Last_lvl(game_forScreen));
        bg_m.stop();

  }
```

Так же метод for_select_resume() вызывается после нажатия кнопки в главном меню «Продолжить». Тогда значение curentLvl будет равен значению максимального уровня, до которого дошел игрок.

После выбора уровня вызывается метод установки экрана, в качестве параметра создается и передается экземпляр класса, например Glava1. Зависимость создаваемого и передаваемого в качестве параметра экземпляра класса определяется значением переменной curentLvl. Классы Glava1, Glava2, Glava3, Glava4, Glava5 являются дочерними классами (наследниками) класса LabirintScr. Данные классы не на много отличаются от родительского класса. В них лишь переопределены методы, которые специфичны для данных уровней.

Класс LabirintScr является ключевым классом, в которой описана основная механика игры:

- Управление роботом и его взаимодействие с окружением.
- Вывод на экран уровня, робота и интерфейса для управления.

Сам класс LabirintScr является дочерним классом (наследником) класса ScreenAdapter, что позволяет использовать его наследников как экраны. В кон-

структоре класса происходит инициализация всех переменных, загрузка файлов с текстурами, звуков и шрифтов.

Метод the_main_game вызывается в методе render(). В данном методе описан алгоритм, когда игрок управляет роботом, передвигаясь по лабиринту. В начале, в методе showGP(), проверяется положение робота относительно камеры и если игрок начинает управлять роботом, то камера плавно двигается за роботом. В методе ShowBackground() рисуется задний фон, затем в методе WherePlayer() определяется положение робота в игровом пространстве и в методе PlayerLogic() происходит рисование робота. Так же в методе PlayerLogic() описана логика взаимодействия робота с окружением. Далее в методе ShowM() рисуется сам лабиринт и в методе Radar() рисуется схематический вид лабиринта и положение робота в нем.

```
    void the_main_game(float d) {
        cam.update();
        batch.setProjectionMatrix(cam.combined);
        showGP();
        batch.begin();
        {
            ShowBackground();
            WherePlayer();
            PlayerLogic();
            if(game_forScreen.effectVol>0f) player.TheSound(d, robot_speed,
game_forScreen.effectVol, robot_jump, robot_jump_temp);
            ShowM();
            Radar();
            player.UpdateCord();

        }
        batch.end();
        camHUD.update();
        batch.setProjectionMatrix(camHUD.combined);
        stage.act(Gdx.graphics.getDeltaTime());
        stage.draw();
    }
```

Как было сказано ранее, в методе ShowM() происходит вывод лабиринта на экран. На самом деле в данном методе реализован алгоритм для вывода лишь той части лабиринта, которая видима для игрока. Данный метод реализован для повышения производительности, т.к. не нужно отображать на экран то, чего игрок не видит. Если на экран выводить весь лабиринт, то чем больше лабиринт, тем больше понадобится системных ресурсов устройства, что критично

для бюджетных мобильных устройств. Так же этот метод наглядно показывает процесс работы технологии тайловой (плиточной) графики.

```
    void ShowM() {

        int nx = 0, ny = 0, ex = 12, ey = 12, dx = 0, dy = 0;
        if (pl_x > 6) {
            nx = pl_x - 6;
            ex = pl_x + 6;
        }
        if (pl_y > 6) {
            ny = pl_y - 6;
            ey = pl_y + 6;
        }
        if (pl_x > ww + ww - 6) {
            nx = ww + ww - 12;
            ex = ww + ww + 1;
        }
        if (pl_y > hh + hh - 6) {
            ny = hh + hh - 12;
            ey = hh + hh + 1;
        }
        for (dx = 0; nx + dx < ex; dx++) {
            for (dy = 0; ny + dy < ey; dy++) {
                if (lab[nx + dx][ny + dy] == 4)
                    batch.draw(walls.findRegion("4"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 31)
                    batch.draw(walls.findRegion("31"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 32)
                    batch.draw(walls.findRegion("34"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 33)
                    batch.draw(walls.findRegion("33"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 34)
                    batch.draw(walls.findRegion("32"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 51)
                    batch.draw(walls.findRegion("51"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 52)
                    batch.draw(walls.findRegion("54"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 53)
                    batch.draw(walls.findRegion("53"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 54)
                    batch.draw(walls.findRegion("52"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 61)
                    batch.draw(walls.findRegion("62"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 62)
                    batch.draw(walls.findRegion("61"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 63)
                    batch.draw(walls.findRegion("64"), (nx + dx) * CellSize, (ny
```

```
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 64)
                    batch.draw(walls.findRegion("63"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 21)
                    batch.draw(walls.findRegion("21"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 22)
                    batch.draw(walls.findRegion("22"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);

                else if (lab[nx + dx][ny + dy] == 27)
                    batch.draw(walls.findRegion("27"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 26)
                    batch.draw(walls.findRegion("26"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 25)
                    batch.draw(walls.findRegion("25"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);

                else if (lab[nx + dx][ny + dy] == 71)
                    batch.draw(walls.findRegion("73"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 72)
                    batch.draw(walls.findRegion("72"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 73)
                    batch.draw(walls.findRegion("71"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 74)
                    batch.draw(walls.findRegion("74"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 11)
                    batch.draw(walls.findRegion("12"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 12)
                    batch.draw(walls.findRegion("11"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 13)
                    batch.draw(walls.findRegion("14"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 14)
                    batch.draw(walls.findRegion("13"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 10)
                    batch.draw(walls.findRegion("16"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 15)
                    batch.draw(walls.findRegion("15"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 16)
                    batch.draw(walls.findRegion("10"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
                else if (lab[nx + dx][ny + dy] == 17)
                    batch.draw(walls.findRegion("17"), (nx + dx) * CellSize, (ny
+ dy) * CellSize);
            }
        }
    }
```

В классах, наследуемых от ScreenAdapter, переопределяется и реализовывается метод dispose(). После завершения игры, закрытия приложения, вызывается метод dispose(). В данном методе освобождаются (удаляются) все ресурсы которые использовались в игре, например текстуры, звуки, сцены и т.д.

```
@Override
public void dispose() {
    batch.dispose();
    walls.dispose();
    ui_atl.dispose();
    stage.dispose();
    gfont.dispose();
    q_atl.dispose();
    q_stage1.dispose();
    q_stage2.dispose();
    bg_music.dispose();
    k_stage.dispose();
    k_atl.dispose();
    btn_b.dispose();
    bbbg.dispose();
    player.dispose();
}
```

Так же в классе LabirintScr объявляются и создаются экземпляры классов ByPlayer и Emaze. Класс ByPlayer представляет собой модель игрока. В нем реализованы необходимые методы для анимации робота, методы для звуковых эффектов издаваемых роботом, переменные необходимые для определения робота в пространстве и характера взаимодействия с окружающими объектами.

Класс Emaze представляет собой модель лабиринта. В нем реализован метод для генерации лабиринта алгоритмом Эллера. Алгоритм Эллера позволяет создавать лабиринты, имеющие только один путь между двумя точками. Сам по себе алгоритм очень быстр и использует память эффективнее, чем другие популярные алгоритмы (такие как Прима и Крускала), требуя памяти пропорционально числу строк. Это позволяет создавать лабиринты большого размера при ограниченных размерах памяти.

```
public void GeneratMaze(){
    int x,y=0,i=1,r;
    Array<Integer> xx = new Array<Integer>();
    for(x=0;x<ww;x++,i++)
        if (x==ww-1) mazeE[x][y]=new CellMaze(true,false,i);
        else mazeE[x][y]=new CellMaze(false,false,i);
    for(x=0;x<ww;x++){
        r=MathUtils.random(1);
        if (r==1) mazeE[x][y].rw=true;
```

```
            else if (r==0 && x!=ww-1) mazeE[x+1][y].cs=mazeE[x][y].cs;
        }
        for(x=0;x<ww;x++){
            if (x==0 && !mazeE[x][y].rw)
                xx.add(x);
            else if (x!=ww-1 && !mazeE[x][y].rw) {
                xx.add(x);
            }
            else if (x!=0 && mazeE[x][y].rw && !mazeE[x-1][y].rw){
                xx.add(x); i=xx.size;
                for(;i>1;i--){
                    r=MathUtils.random(i-1);
                    mazeE[xx.removeIndex(r)][y].dw=true;
                }
                xx.clear();
            }
            else if (x!=0 && mazeE[x][y].rw && mazeE[x-1][y].rw) xx.clear();
        }
        for(x=1;x<ww;x++){
            for(y=1;y<hh;y++){
                if (y!=hh-1){
                    for(x=0;x<ww;x++) mazeE[x][y]=
new CellMaze(mazeE[x][y-1].rw,mazeE[x][y-1].dw,mazeE[x][y-1].cs);
                    for(x=0;x<ww;x++){
                        if(mazeE[x][y].rw && x!=ww-1) mazeE[x][y].rw=false;
                        if(x==0 && mazeE[x][y].dw) {mazeE[x][y].dw=false;
mazeE[x][y].cs=1;}
                        else if (mazeE[x][y].dw) {mazeE[x][y].dw=false;
mazeE[x][y].cs=mazeE[x-1][y].cs+1;}
                    }
                    for(x=0;x<ww;x++){
                        r=MathUtils.random(1);
                        if (r==1) mazeE[x][y].rw=true;
                        else if (r==0 && x!=ww-1)
mazeE[x+1][y].cs=mazeE[x][y].cs;
                    }
                    for(x=0;x<ww;x++){
                        if (x==0 && !mazeE[x][y].rw)
                            xx.add(x);
                        else if (x!=ww-1 && !mazeE[x][y].rw) {
                            xx.add(x);
                        }
                        else if (x!=0 && mazeE[x][y].rw && !mazeE[x-1][y].rw){
                            xx.add(x); i=xx.size;
                            for(;i>1;i--){
                                r=MathUtils.random(i-1);
                                mazeE[xx.removeIndex(r)][y].dw=true;
                            }
                            xx.clear();}
                        else if (x!=0 && mazeE[x][y].rw && mazeE[x-1][y].rw)
                        xx.clear();}
                }
                else if (y==hh-1){
                    for(x=0;x<ww;x++)
                        if (x==ww-1) mazeE[x][y]=new Cell-
Maze(true,true,mazeE[x][y-1].cs);
                    else mazeE[x][y]=new CellMaze(false,true,mazeE[x][y-1].cs);
                }
            }
        }
}
```

После генерации лабиринта в функции ConvMaze(byte[][] lab) лабиринт преобразовывается в удобный вид для использования его при рисовании лабиринта тайловым методом. Так же в данном классе есть методы для генерации стен с загадками, и входа и выхода из лабиринта.

Каждый раз, проходя лабиринт, переменная curentLvl увеличивается на единицу. Если переменная curentLvl будет равна 50, то игрок попадает на последний уровень (Рисунок 2.2.12).

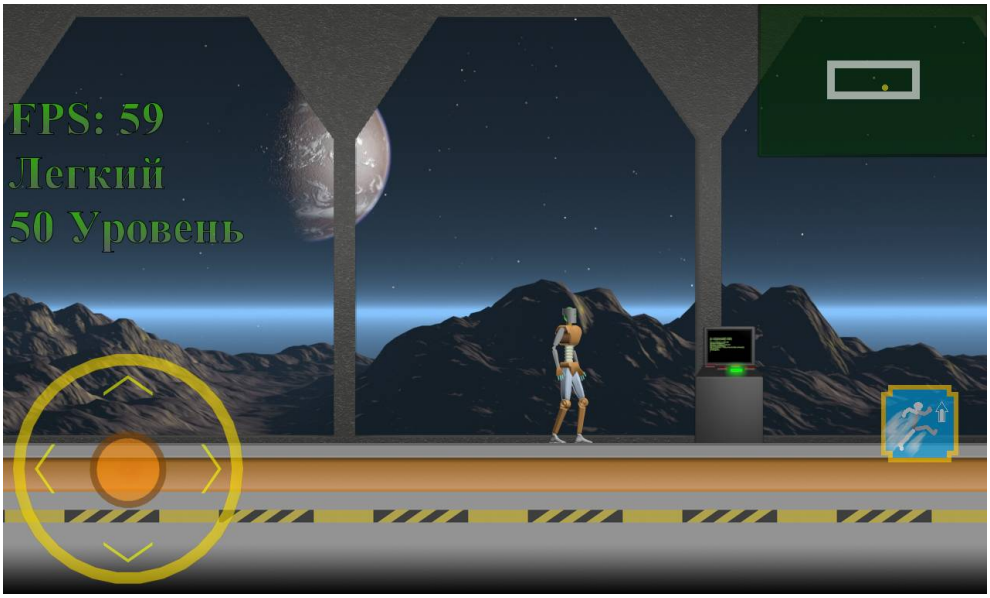


Рисунок 2.2.12. Последний уровень в игре.

В последнем уровне роботу нужно подойти к панели управления и нажать кнопку действия, после чего игра заканчивается титрами на фоне картинки с роботом и фоновой музыкой (Рисунок 2.2.13).

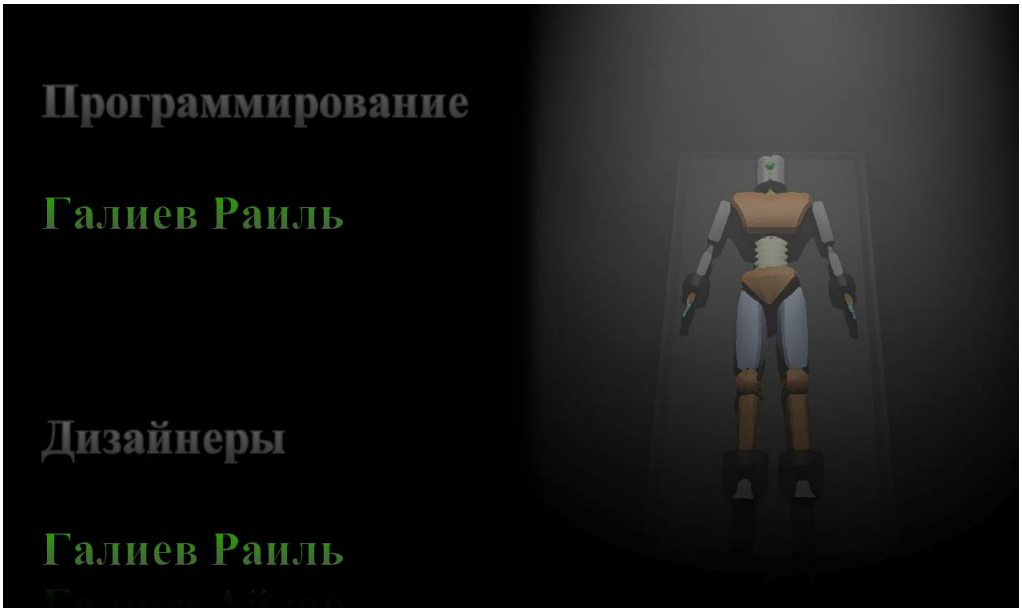


Рисунок 2.2.13. Титры в конце игры.

2.3 Результаты апробации

Игра была опубликована в Google Play 7 сентября 2016 года, распространяется бесплатно. На момент написания данной работы, игру скачали и установили всего 427 человек (Рисунок 2.3.1).

НАЗВАНИЕ ПРИЛОЖЕНИЯ	ЦЕНА	УСТАНОВОК (АКТИВНЫХ/ ВСЕГО)
Проект АРсИИ: Эксперимент 1.1.5	Бесплатное	19 / 427

Рисунок 2.3.1. Количество установок.

В консоли разработчика Google Play (Google Play Developer Console) доступно множество инструментов для анализа опубликованного приложения (Рисунок 2.3.2)

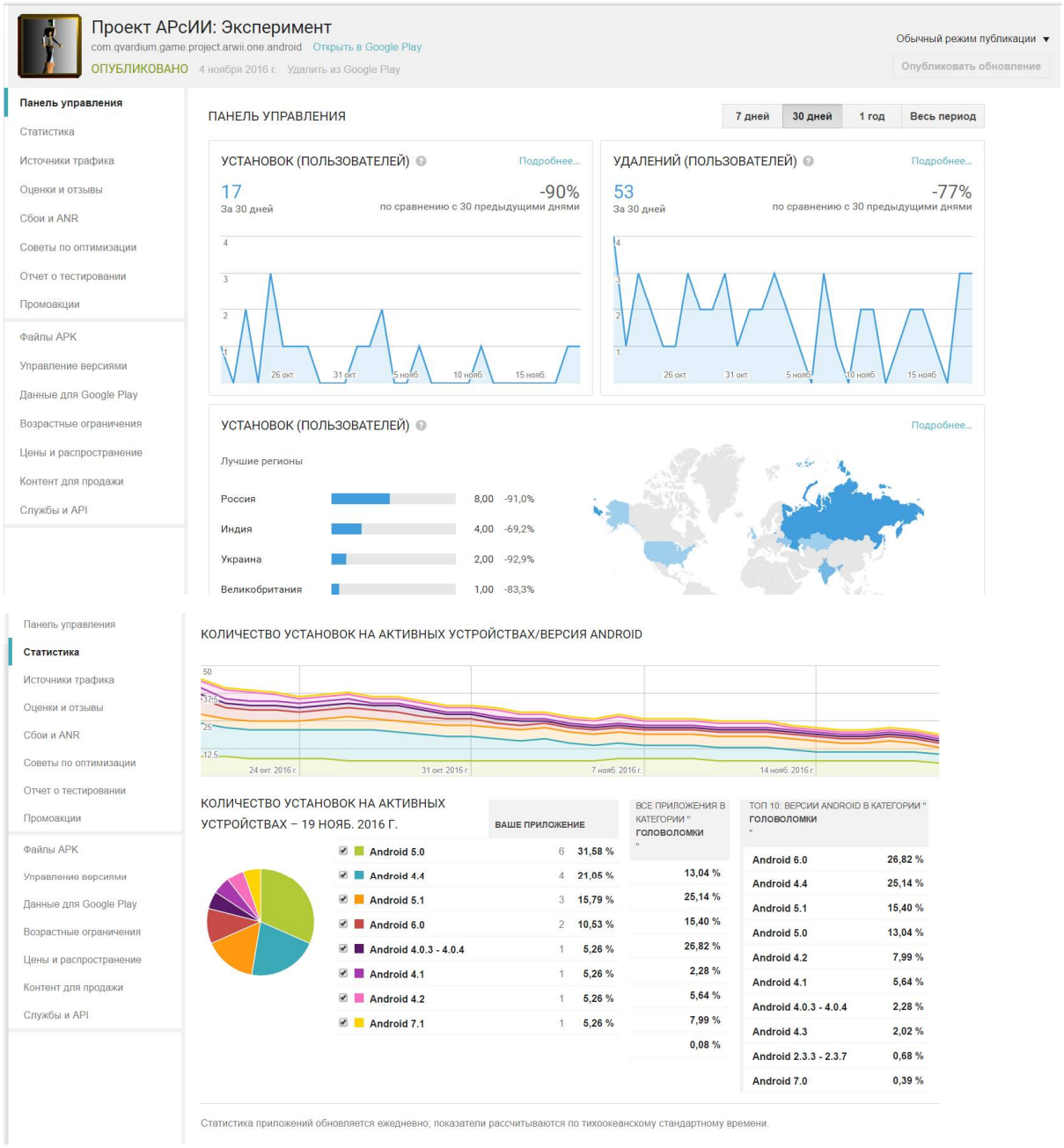


Рисунок 2.3.2. Вид панели управления и статистики в Google Play Developer Console.

По результатам оценок приложение имеет рейтинг 4,25 (Рисунок 2.3.3).

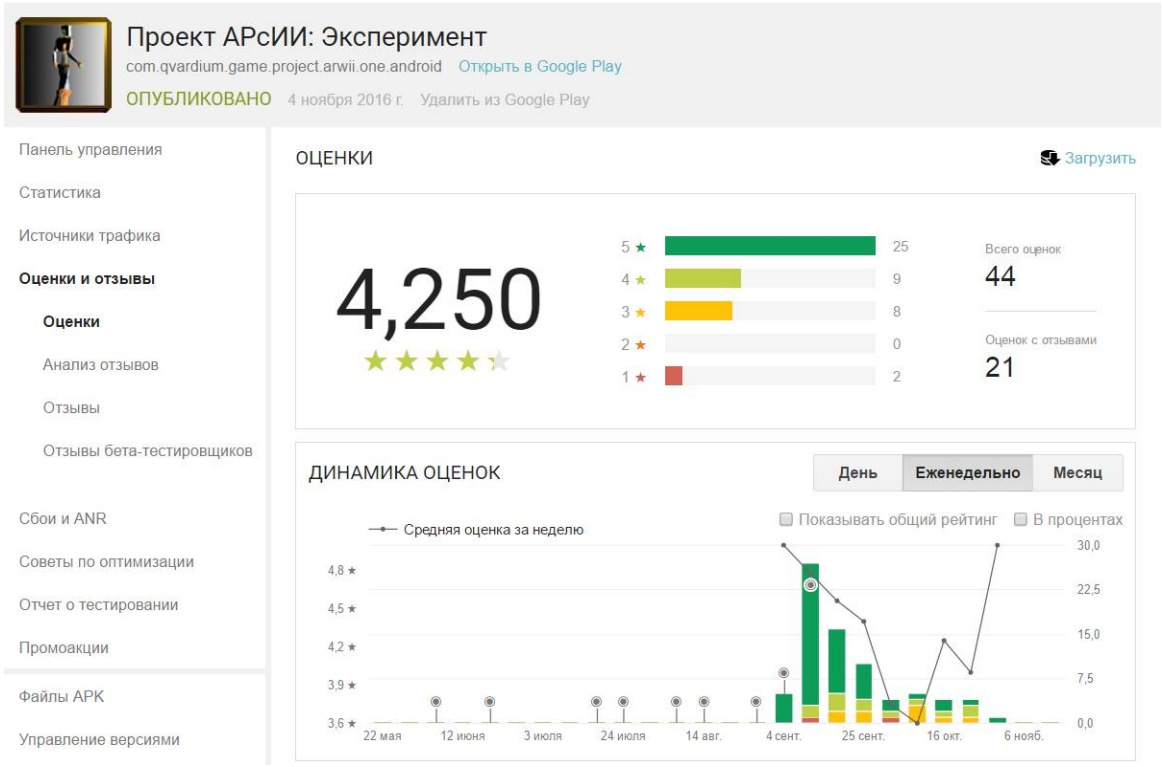
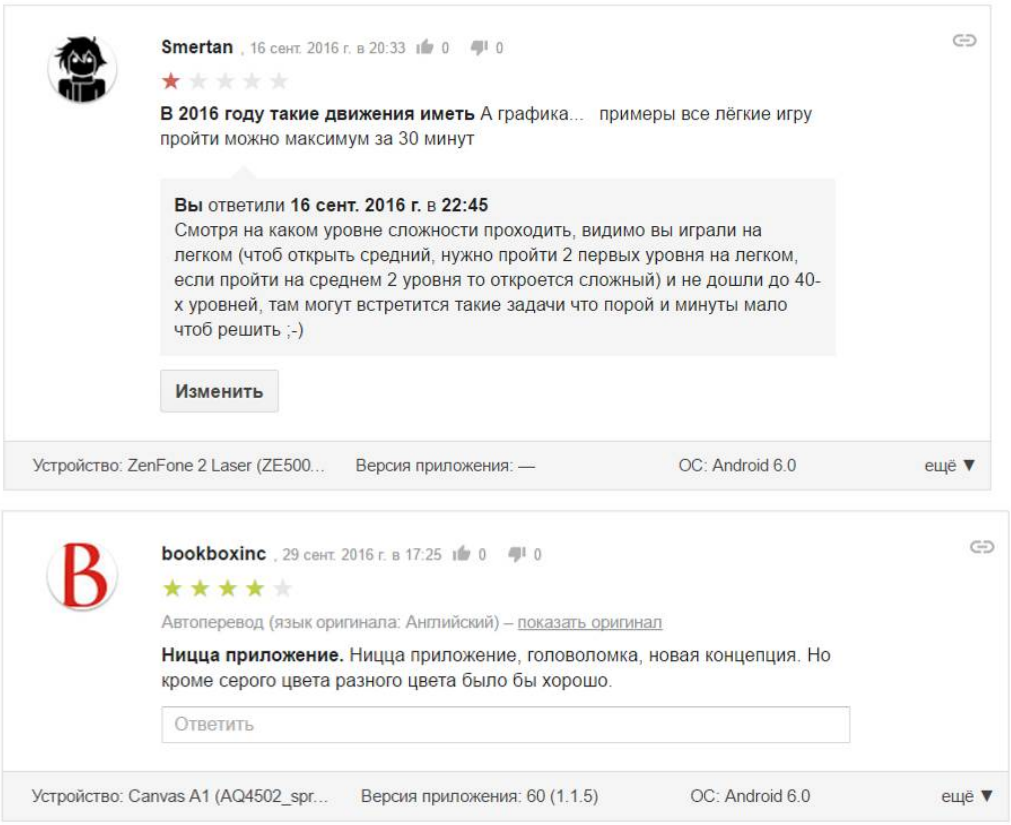


Рисунок 2.3.3. Оценки пользователей.



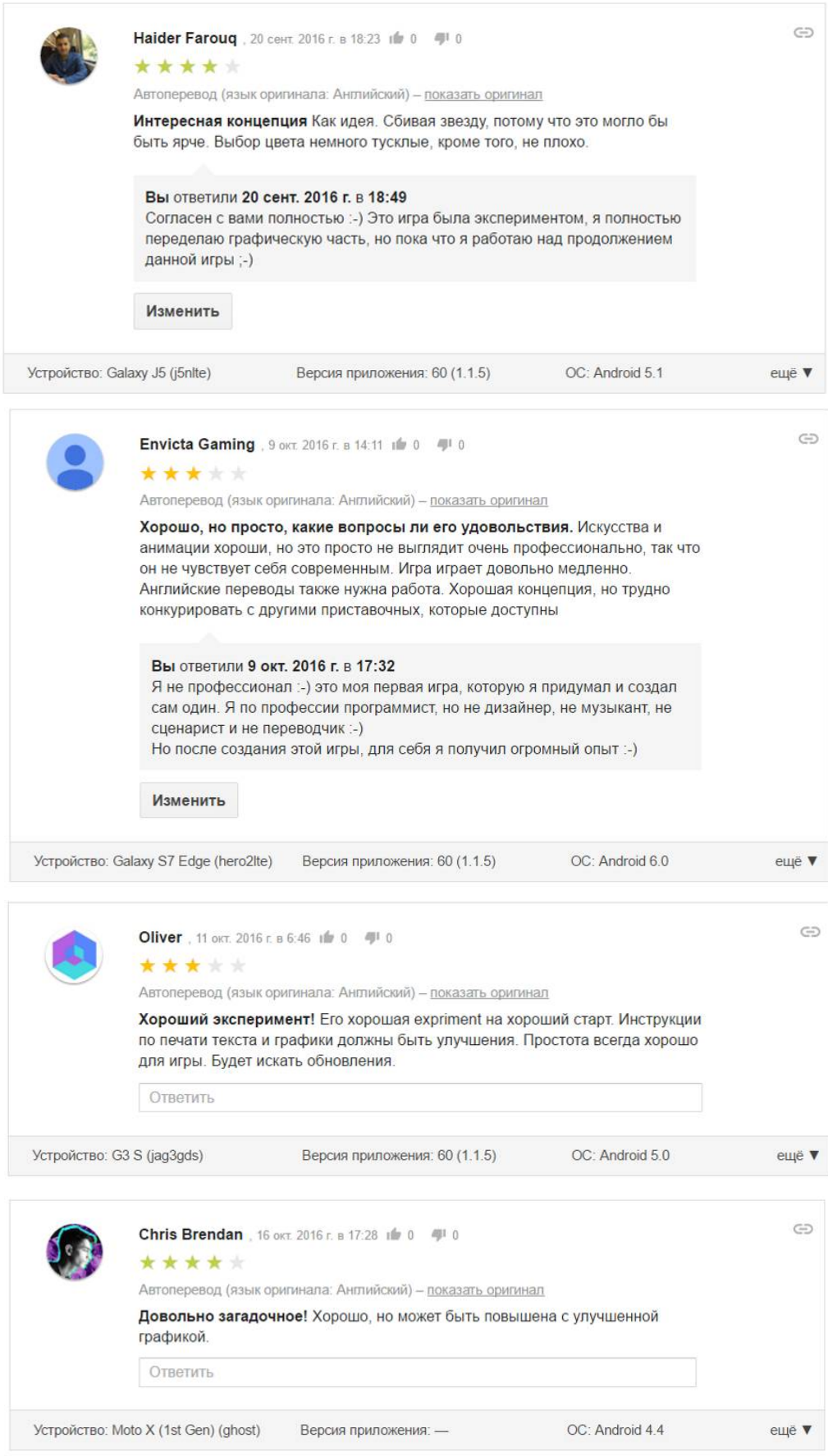


Рисунок 2.3.4. Отзывы некоторых пользователей.

В игре был встроен рекламный банер AdMob и видеореклама с вознаграждением Chartboost. По результатам на сайте AdMob с помощью рекламного банера было заработано 3 цента (Рисунок 2.3.5).

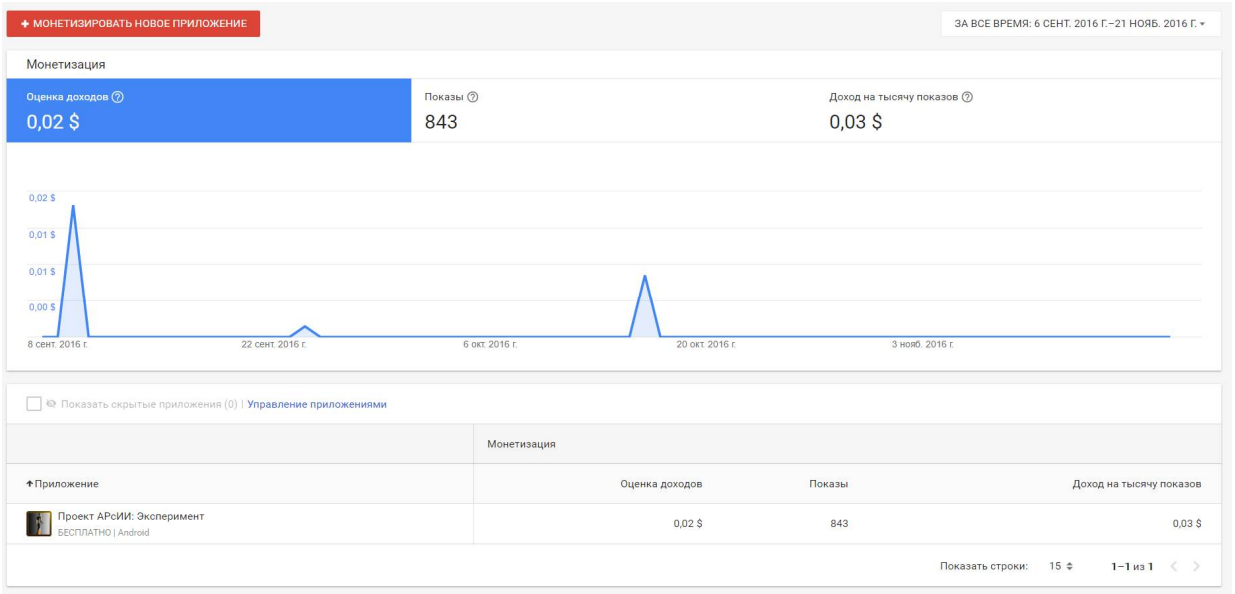


Рисунок 2.3.5. Вид панели статистики рекламной площадки AdMob.

По результатам на сайте Chartboost с помощью видео рекламы с вознаграждением было заработано 0$ (Рисунок 2.3.6).

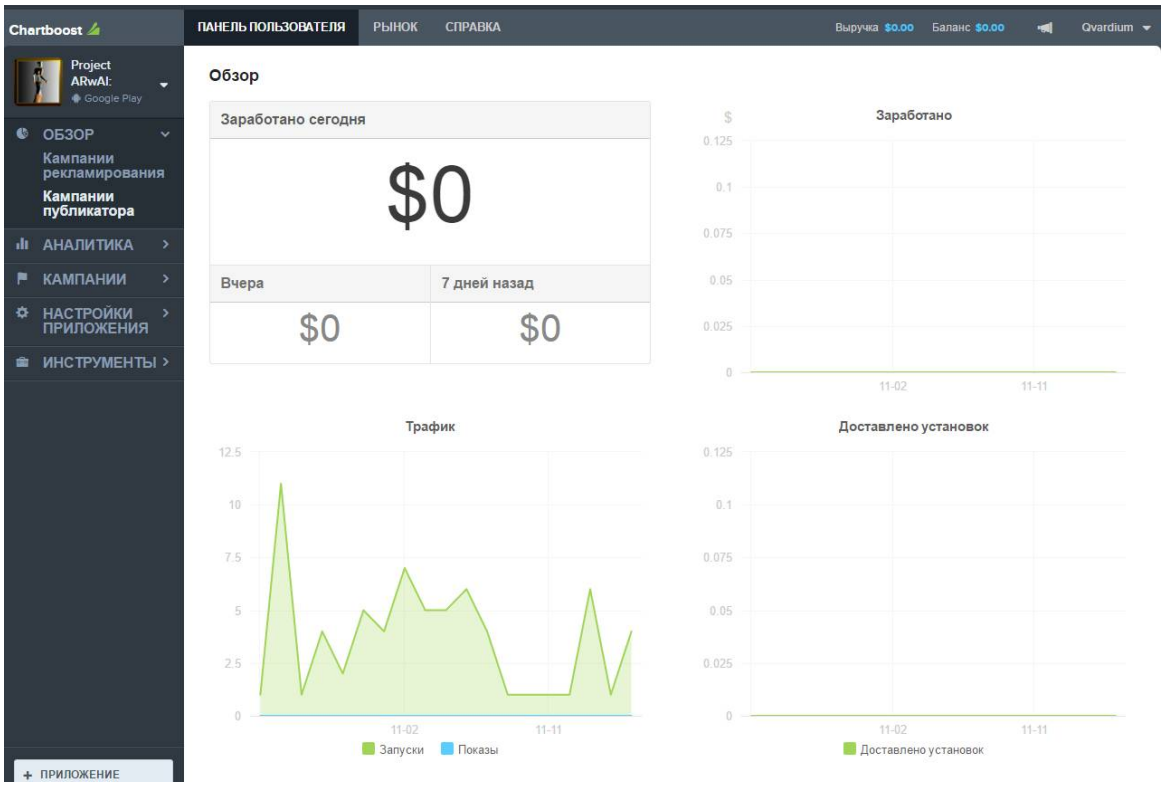


Рисунок 2.3.6. Вид панели статистики рекламной площадки Chartboost.

После публикации игры были обноружены и исправлены ошибки. Всего с момента публикации рабочей версии 1.0.0, было исправлено и выпущено 16 обновлений, последняя на данный момент стабильная версия 1.1.5. Данное игровое приложение поддерживают 12339 устройств.

Анализируя письменные отзывы пользователей (Рисунок 2.3.4) был сделан вывод, что графическое исполнение игры выглядит не актуальным, много серого цвета. Пользователи предлагают:

- улучшить графическую составляющую игры;
- сделать качественный перевод на английский язык;
- изменить игровую механику сделав игровой процесс более динамичным.

Заключение

В результате исследования и анализа операционной системы Android, был сделан вывод, что разработка игровых приложений для данной платформы является актуальной и востребованной. Также было доказано, что с помощью бесплатных и доступных инструментов, как IDE Android Studio и фреймворка libGDX, возможно создавать игровые приложения, которые не требовательны к ресурсам физического устройства.

В ходе исследования:

1. Проанализированы особенности разработки приложения для операционной системы Android.
2. Произведен подробный анализ инструмента для разработки приложений IDE Android Studio и обоснован выбор технологии разработки приложений с использованием фреймворка libGDX.
3. В соответствии с техническим заданием было разработано и протестировано (на реальных устройствах) игровое приложение «Проект АРсИИ: Эксперимент» для операционной системы Android.
4. Разработанное игровое приложение было опубликовано в Google Play и собраны статистические данные из: Google Play Developer Console, AdMob и Chartboost. На основе статистических данных произведен анализ отзывов пользователей в Google Play Developer Console и анализ доходов с рекламных площадок AdMob и Chartboost.

Библиографический список

1. Дейтел П., Дейтел Х., Дейтел Э. Android для разработчиков. 2-е изд. СПб.: Питер, 2015. 384 с.
2. Марио Ц. Программирование игр под Android. СПб.: Питер, 2013. 688 с.
3. Детальный анализ Android RunTime (ART) в Android L // «Хакер» - Безопасность, разработка, DevOps URL: https://xakep.ru/2014/07/03/art-vm/ (дата обращения: 21.11.2016).
4. David Saltares Márquez, Alberto Cejas Sánchez Libgdx Cross-platform Game Development Cookbook. Birmingham B3 2PB, UK.: Packt Publishing Ltd, 2014. 516 c.
5. Indraneel Potnis. LibGDX Cross-Platform Development Blueprints. Birmingham B3 2PB, UK.: Packt Publishing Ltd, 2015. 316 c.
6. Sebastián Di Giuseppe, Andreas Krühlmann, Elmar van Rijnswou. Building a 3D Game with LibGDX. Birmingham B3 2PB, UK.: Packt Publishing Ltd, 2016. 227 c.
7. История Android // Android – История URL: https://www.android.com/intl/ru_ru/history/#/marshmallow (дата обращения: 21.11.2016).
8. Этапы создания компьютерной игры // Этапы создания компьютерной игры (Сайт GamesisArt.ru) URL: http://gamesisart.ru/game_dev_create.html (дата обращения: 21.11.2016).
9. About // About | Box2D URL: http://box2d.org/about/ (дата обращения: 21.11.2016).
10. Android Studio // Android Studio — Википедия URL: https://ru.wikipedia.org/wiki/Android_Studio (дата обращения: 21.11.2016).
11. Dashboards // Dashboards | Android Developers URL: https://developer.android.com/about/dashboards/index.html (дата обращения: 15.10.2016).

12. FreeType // The FreeType Project URL: https://www.freetype.org/ (дата обращения: 21.11.2016).
13. Kiss FFT // Kiss FFT download | SourceForge.net URL: https://sourceforge.net/projects/kissfft/ (дата обращения: 21.11.2016).
14. libGDX - фреймворк для разработки игр // libGDX: libGDX - фреймворк для разработки игр URL: http://www.libgdx.ru/2013/08/introduction.html (дата обращения: 21.11.2016).
15. Meet Android Studio // Meet Android Studio | Android Studio URL: https://developer.android.com/studio/intro/index.html#the_user_interface (дата обращения: 21.11.2016).
16. Mobile application protection // Mobile application protection | GuardSquare URL: https://www.guardsquare.com/en/mobile-application-protection (дата обращения: 21.11.2016).
17. mpg123 - Fast console MPEG Audio Player and decoder library // mpg123, Fast MP3 Player for Linux and UNIX systems URL: http://mpg123.org/ (дата обращения: 21.11.2016).
18. Nextpeer Social // Nextpeer Social - Nextpeer Social - Nextpeer Knowledgebase URL: https://nextpeer.atlassian.net/wiki (дата обращения: 21.11.2016).
19. OpenGL Overview // OpenGL Overview URL: https://www.opengl.org/about/ (дата обращения: 21.11.2016).
20. SoundTouch Audio Processing Library // SoundTouch Audio Processing Library URL: http://www.surina.net/soundtouch/ (дата обращения: 21.11.2016).
21. vorbis.com // Vorbis.com URL: http://vorbis.com/ (дата обращения: 21.11.2016).
22. What is LWJGL 3? // LWJGL - Lightweight Java Game Library URL: https://www.lwjgl.org/ (дата обращения: 21.11.2016).
23. What is OpenAL? // OpenAL: Cross Platform 3D Audio URL: http://www.openal.org/ (дата обращения: 21.11.2016).
24. What is Spine? // Spine: In Depth URL: http://ru.esotericsoftware.com/spine-in-depth#What-is-Spine (дата обращения: 21.11.2016).